

Raymond BISDORFF

Algorithmic Decision Making with Python Resources

Tutorials and Advanced Topics

July 4, 2021

Springer

Contents

Part I Introduction to the DIGRAPH3 software resources

1	Working with the DIGRAPH3 Python resources	3
1.1	Installing the DIGRAPH3 resources	3
1.2	Starting a Python3 session	4
1.3	Permanent storage of a digraph object	5
1.4	Inspecting a digraph object	7
1.5	Special digraph instances	11
2	Working with bipolar-valued digraphs	13
2.1	Random bipolar-valued digraphs	13
2.2	Graphviz drawings	15
2.3	Asymmetric and symmetric parts	16
2.4	Border and inner parts	17
2.5	Fusion by epistemic disjunction	19
2.6	Dual, converse and codual digraphs	19
2.7	Symmetric and transitive closures	21
2.8	Strong components	22
2.9	CSV storage	23
2.10	Complete, empty and indeterminate digraphs	24
3	Working with outranking digraphs	27
3.1	Outranking digraph model	27
3.2	The bipolar-valued outranking digraph	29
3.3	Pairwise comparisons	30
3.4	Recoding the digraph valuation	31
3.5	The strict outranking digraph	32

Part II Evaluation models and decision methods and tools

4	How to create a new performance tableau instance	37
4.1	Editing a template file	37
4.2	Editing the decision alternatives	39
4.3	Editing the decision objectives	40
4.4	Editing the family of performance criteria	41
4.5	Editing the performance table	43
4.6	Inspecting the template outranking relation	45
4.7	Ranking the template peformance tableau	46
5	Generating random performance tableaux	49
5.1	Introduction	49
5.2	Random standard performance tableaux	50
5.3	Random Cost-Benefit performance tableaux	52
5.4	Random three objectives performance tableaux	56
5.5	Random academic performance tableaux	59
5.6	Random linearly ranked performance tableaux	62
6	Computing a best choice recommendation	63
6.1	What site to choose ?	63
6.2	The DIGRAPH3 Performance tableau	65
6.3	Computing the outranking digraph	67
6.4	Computing a RUBIS best choice recommendation	69
6.5	Computing <i>strict best</i> choice recommendations	71
6.6	Weakly ordering the outranking digraph	73
7	Computing the winner of an election	77
7.1	Linear voting profiles	77
7.2	Computing the winner	78
7.3	The Condorcet winner	80
7.4	Cyclic social preferences	81
7.5	On generating realistic random linear voting profiles	83
8	Ranking with multiple incommensurable criteria	89
8.1	The ranking problem	89
8.2	The COPELAND ranking	91
8.3	The NETFLOWS ranking	93
8.4	KEMENY rankings	95
8.5	SLATER rankings	97
8.6	KOHLER's ranking-by-choosing rule	99
8.7	Tideman's RANKEDPAIRS ranking rule	101
9	Rating by sorting into relative performance quantiles	103
9.1	Quantile sorting on a single criterion	103
9.2	Quantiles sorting with multiple performance criteria	104
9.3	The sparse pre-ranked outranking digraph model	108
9.4	Ranking pre-ranked sparse outranking digraphs	111

Contents	vii
----------	-----

10 Rating by ranking with learned performance quantile norms	113
10.1 The absolute rating problem	113
10.2 Incremental learning of historical performance quantiles	114
10.3 Rating-by-ranking new performances with quantile norms	116
11 HPC ranking of big performance tableaux	125
11.1 C-compiled Python modules	125
11.2 Big Data performance tableaux	126
11.3 C-implemented integer-valued outranking digraphs	127
11.4 The sparse outranking digraph implementation	129
11.5 Ranking big performance tableaux	132
11.6 HPC quantiles ranking records	135

Part III Evaluation and decision case studies

12 Alice's best choice: A selection case study	139
12.1 The decision problem	139
12.2 The performance tableau	141
12.3 Building a best choice recommendation	143
12.4 Robustness analysis	148
13 The best academic Computer Science Depts: A ranking case study	153
13.1 The THE performance tableau	153
13.2 Ranking with multiple criteria of ordinal significance	159
13.3 How to judge the quality of a ranking result?	166
14 The best students, where do they study? A rating case study	173
14.1 The performance tableau	173
14.2 Rating-by-ranking with lower-closed quantile limits	177
14.3 Inspecting the bipolar-valued outranking digraph	181
14.4 Rating by quantiles sorting	182
15 Exercises	189
15.1 Who will receive the best student award? (\$)	189
15.1.1 Data	189
15.1.2 Questions	190
15.2 How to fairly rank movies (\$)	190
15.2.1 Data	190
15.2.2 Questions	191
15.3 What is your best choice recommendation? (\$)	192
15.3.1 Data Footnote[43]	192
15.3.2 Questions	193
15.4 What is the best public policy? (\$\$)	193
15.4.1 Data files	193
15.4.2 Questions	193
15.5 A fair diploma validation decision (\$\$\$\$)	194

15.5.1 Data	194
15.5.2 Questions	194

Part IV Advanced topics

16 Ordinal Correlation	197
16.1 Coping with missing data	197
16.2 Modelling pairwise bipolar-valued rating opinions	199
16.3 KENDALL's tau index	202
16.4 Bipolar-valued relational equivalence	203
16.5 Fitness of ranking heuristics	205
16.6 Illustrating preference divergences	207
16.7 Exploring the “ <i>better rated</i> ” and the “ <i>as well as rated</i> ” opinions ..	209
17 On computing digraph kernels	211
17.1 What is a graph kernel ?	211
17.2 Initial and terminal kernels	213
17.3 Kernels in lateralized digraphs	218
17.4 Computing first and last choice recommendations	221
17.5 Tractability of prekernel computation	224
17.6 BERGE kernel equation systems	226
18 On confident outrankings	235
18.1 Modelling uncertain criteria significances	235
18.2 Bipolar-valued likelihood of “ <i>at least as good as</i> ” situations	237
18.3 Confidence level of outranking situations	239
19 On robust outrankings	245
19.1 Cardinal or ordinal criteria significances	245
19.2 Qualifying the stability of outranking situations	246
19.3 Computing stability denotations	250
19.4 Robust bipolar-valued outranking digraphs	252
19.5 Unopposed outranking situations	255
19.6 Computing Pareto efficient multiobjective choices	258
20 Two-stage elections with multipartisan primary selection	261
20.1 Converting voting profiles into performance tableaux	261
20.2 Multipartisan primary selection of eligible candidates	263
20.3 Secondary election winner determination	265
20.4 Multipartisan preferences in divisive politics	265
21 Bipolar approval voting	269
21.1 Bipolar approval voting systems	269
21.2 Pairwise comparison of bipolar approval votes	272
21.3 Three-valued evaluative voting system	274
21.4 Favouring multipartisan candidates	277

Part V Working with simple graphs

22 Working with undirected graphs	285
22.1 Implementing undirected graphs	285
22.2 q-coloring of a graph	288
22.3 MIS and clique enumeration	289
22.4 Line graphs and maximal matchings	290
22.5 Grids and the Ising model	293
22.6 Simulating Metropolis random walks	294
23 On tree graphs and graph forests	297
23.1 Generating random tree graphs	297
23.2 Recognizing tree graphs	299
23.3 Spanning trees and forests	301
23.4 Maximum determined spanning forests	303
24 Computing the non isomorphic MISs of the n-cycle graph	307
24.1 Computing maximal independent sets (MISs)	308
24.2 Computing the automorphism group	309
24.3 Computing the isomorphic MISs	310
25 About split, interval and permutation graphs	313
25.1 A multiply perfect graph	315
25.2 Who is the liar?	317
25.3 Generating permutation graphs	319
25.4 Recognizing permutation graphs	322

Part I

**Introduction to the DIGRAPH3 software
resources**

The first part contains a set of tutorials introducing the DIGRAPH3 software collection of Python3 modules.

The basic idea of the DIGRAPH3 Python resources is to make easy python interactive sessions or write short Python scripts for computing all kind of results from a bipolar-valued digraph or graph. These include such features as maximal independent, maximal dominant or absorbent choices, rankings, outrankings, linear ordering, etc. Most of the available computing resources are meant to illustrate a Master Course on Algorithmic Decision Theory given at the University of Luxembourg in the context of its Master in Information and Computer Science (MICS).

The Python development of these computing resources offers the advantage of an easy to write and maintain OOP source code as expected from a performing scripting language without loosing on efficiency in execution times compared to compiled languages such as C++ or Java.

Chapter 1

Working with the DIGRAPH3 Python resources

Abstract The basic idea of the DIGRAPH3 Python resources is to make easy python interactive sessions or write short Python scripts for computing all kind of results from a bipolar-valued digraph or graph. These include such features as maximal independent, maximal dominant or absorbent choices, rankings, outrankings, linear ordering, etc. Most of the available computing resources are meant to illustrate a Master Course on Algorithmic Decision Theory given at the University of Luxembourg in the context of its Master in Information and Computer Science (MICS). The Python development of these computing resources offers the advantage of an easy to write and maintain OOP source code as expected from a performing scripting language without loosing on efficiency in execution times compared to compiled languages such as C++ or Java.

1.1 Installing the DIGRAPH3 resources

Using the DIGRAPH3 modules is easy. You only need to have installed on your system the Python programming language of version 3.+ (readily available under Linux and Mac OS). Notice that, from Version 3.3 on, the Python standard decimal module implements very efficiently its decimal.Decimal class in C. Now, Decimal objects are mainly used in the DIGRAPH3 characteristic r-valuation functions, which makes the recent python-3.7+ versions much faster (more than twice as fast) when extensive digraph operations are performed.

Several download options (easiest under Linux or Mac OS-X) are given. Either, by using a git client either, from [github](https://github.com/rbisdorff/Digraph3):

```
...\$ git clone https://github.com/rbisdorff/Digraph3
```

Or, from sourceforge.net:

```
...\$ git clone https://git.code.sf.net/p/digraph3/code Digraph3
```

Or, with a browser access, download either, from the [github](https://github.com/rbisdorff/Digraph3) link above or, from the [sourceforge](http://sourceforge.net) page the latest distribution zip archive and extract it. On Linux or Mac OS, ..\$ cd to the extracted Digraph3 directory.

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

installs (with *sudo* !!) the DIGRAPH3 modules in the current running python environment. Python 3.8 (or later) environment is recommended (see the makefile for adapting to your python environment). Whereas:

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

installs the DIGRAPH3 modules in an activated virtual python environment. If the **cython** (<https://cython.org>) C-compiled modules for Big Data applications are required, it is necessary to previously install the *Cython* package in the running Python environment:

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

It is recommended to run a test suite:

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

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

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

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

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

When the GNU parallel shell tool (<https://www.gnu.org/software/parallel>) is installed and multiple cores are detected, the tests may be executed in multiple processing mode:

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

Individual module `pytest` suites are also provided (see the makefile), like the one for the `outrankingDigraphs` module:

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

Dependencies

- To be fully functional, the DIGRAPH3 resources mainly need the `graphviz` tools (<https://graphviz.org>) and the R statistics resources (<https://www.r-project.org>) to be installed.
- When exploring digraph isomorphisms, the `nauty isomorphism testing program` is required.
- Two specific criteria and actions clustering methods of the `OutrankingDigraph` class furthermore require the `calmat` matrix computing resource to be installed (see the `calmat` directory in the DIGRAPH3 resources).

1.2 Starting a Python3 session

You may start an interactive Python3 session in the `Digraph3` directory.

```
$HOME/.../Digraph3$ python3
Python 3.9.5 (default, Jun 5 2021, 14:26:42)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or
"license" for more information.
>>>
```

For exploring the classes and methods provided by the DIGRAPH3 modules (see the [DIGRAPH3 documentation](#)) enter the Python3 commands following the session prompts marked with ' ' or '...'. The lines without the prompt are output from the Python3 interpreter.

Listing 1.1 Generating a digraph instance

```
1 >>> from randomDigraphs import RandomDigraph
2 >>> dg = RandomDigraph(order=5,arcProbability=0.5,seed=101)
3 >>> dg
4 *----- Digraph instance description -----*
5 Instance class      : RandomDigraph
6 Instance name       : randomDigraph
7 Digraph Order       : 5
8 Digraph Size        : 12
9 Valuation domain   : [-1.00; 1.00]
10 Determinateness    : 100.000
11 Attributes         : ['actions','valuationdomain',
12                      'relation','order','name',
13                      'gamma','notGamma']
```

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

1.3 Permanent storage of a digraph object

We may save the content of `dg` in a file called `tutorialDigraph.py`.

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

with the following content:

Listing 1.2 A stored digraph instance

```
1 from decimal import Decimal
2 from collections import OrderedDict
3
4 actions = OrderedDict([
5     ('a1', {'shortName': 'a1',
6             'name': 'random decision action'}),
7     ('a2', {'shortName': 'a2',
8             'name': 'random decision action'}),
```

```

9      ('a3', {'shortName': 'a3',
10         'name': 'random decision action'}),
11      ('a4', {'shortName': 'a4',
12         'name': 'random decision action'}),
13      ('a5', {'shortName': 'a5',
14         'name': 'random decision action'}),
15  ])
16
17 valuationdomain = {
18     'min': Decimal('-1.0'),
19     'med': Decimal('0.0'),
20     'max': Decimal('1.0'),
21     'hasIntegerValuation': True, # representation format
22 }
23
24 relation = {
25     'a1': {'a1':Decimal('-1.0'), 'a2':Decimal('-1.0'),
26             'a3':Decimal('1.0'), 'a4':Decimal('-1.0'),
27             'a5':Decimal('-1.0'),},
28     'a2': {'a1':Decimal('1.0'), 'a2':Decimal('-1.0'),
29             'a3':Decimal('-1.0'), 'a4':Decimal('1.0'),
30             'a5':Decimal('1.0'),},
31     'a3': {'a1':Decimal('1.0'), 'a2':Decimal('-1.0'),
32             'a3':Decimal('-1.0'), 'a4':Decimal('1.0'),
33             'a5':Decimal('-1.0'),},
34     'a4': {'a1':Decimal('1.0'), 'a2':Decimal('1.0'),
35             'a3':Decimal('1.0'), 'a4':Decimal('-1.0'),
36             'a5':Decimal('-1.0'),},
37     'a5': {'a1':Decimal('1.0'), 'a2':Decimal('1.0'),
38             'a3':Decimal('1.0'), 'a4':Decimal('-1.0'),
39             'a5':Decimal('-1.0'),},
40 }

```

All digraphs instances are of Digraph type and contain at least the following attributes (see Listing 1.1 Lines 11-12):

1. A name attribute, holding usually the actual name of the stored instance that was used to create the instance;
2. A ordered dictionary of digraph nodes called actions (decision alternatives) with at least a name attribute;
3. An order attribute containing the number of graph nodes (length of the actions dictionary) automatically added by the object constructor;
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);
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;

6. Its associated `gamma` attribute, a dictionary containing the direct successors, respectively predecessors of each action, automatically added by the object constructor;
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.

1.4 Inspecting a digraph object

Different `show` methods (see Listing 1.3 Lines 3, 18, 28,31 below) reveal us now that `dg` is a crisp, irreflexive and connected digraph of order five.

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'}),
6   ('a2', {'shortName': 'a2', 'name': 'random decision action'}),
7   ('a3', {'shortName': 'a3', 'name': 'random decision action'}),
8   ('a4', {'shortName': 'a4', 'name': 'random decision action'}),
9   ('a5', {'shortName': 'a5', 'name': 'random decision action'})
10  ])
11 Valuation domain : {
12   'min': Decimal('-1.0'),
13   'max': Decimal('1.0'),
14   'med': Decimal('0.0'), 'hasIntegerValuation': True
15   }
16
17 >>> dg.showRelationTable()
18 * ----- Relation Table -----
19   S | 'a1'  'a2'  'a3'  'a4'  'a5'
20   -----|-----
21   'a1' | -1    -1    1     -1    -1
22   'a2' |  1     -1    -1     1     1
23   'a3' |  1     -1    -1     1    -1
24   'a4' |  1      1     1     -1    -1
25   'a5' |  1      1     1     -1    -1
26 Valuation domain: [-1;+1]
27
28 >>> dg.showComponents()
29 *--- Connected Components ---*
30 1: ['a1', 'a2', 'a3', 'a4', 'a5']
31
32 >>> dg.showNeighborhoods()
33 Neighborhoods:
34   Gamma   :
35   'a1': in => {'a2', 'a4', 'a3', 'a5'}, out => {'a3'}
36   'a2': in => {'a5', 'a4'}, out => {'a1', 'a4', 'a5'}

```

```

37 'a3': in => {'a1', 'a4', 'a5'}, out => {'a1', 'a4'}
38 'a4': in => {'a2', 'a3'}, out => {'a1', 'a3', 'a2'}
39 'a5': in => {'a2'}, out => {'a1', 'a3', 'a2'}
40     Not Gamma :
41 'a1': in => set(), out => {'a2', 'a4', 'a5'}
42 'a2': in => {'a1', 'a3'}, out => {'a3'}
43 'a3': in => {'a2'}, out => {'a2', 'a5'}
44 'a4': in => {'a1', 'a5'}, out => {'a5'}
45 'a5': in => {'a1', 'a4', 'a3'}, out => {'a4'}

```

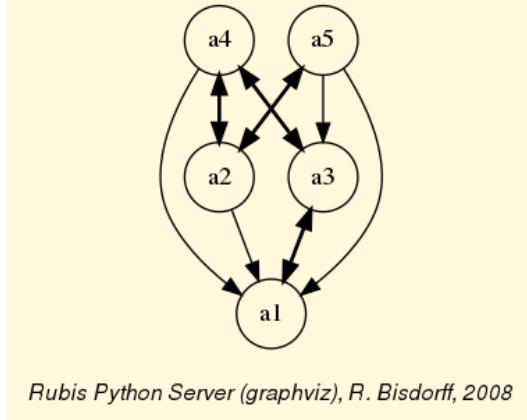
The `exportGraphViz()` method generates in the current working directory a `tutorialDigraph.dot` file and a `tutorialdigraph.png` picture of the tutorial digraph `dg` (see Fig. 1.1), if the `graphviz` tools are installed on your system.

```

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

```

Fig. 1.1 The tutorial crisp digraph. The `exportGraphViz()` method is depending on drawing tools from <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.



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()
2 *---- general statistics -----
3 for digraph : <tutorialDigraph.py>
4 order : 5 nodes
5 size : 12 arcs
6 undetermined : 0 arcs
7 determinateness (%) : 100.0
8 arc density : 0.60
9 double arc density : 0.40
10 single arc density : 0.40

```

```

11 absence density           : 0.20
12 strict single arc density: 0.40
13 strict absence density   : 0.20
14 nbr. of components       : 1
15 nbr. of strong components: 1
16 transitivity degree (%) : 60.0
17                               : [0, 1, 2, 3, 4, 5]
18 outdegrees distribution   : [0, 1, 1, 3, 0, 0]
19 indegrees distribution    : [0, 1, 2, 1, 1, 0]
20 mean outdegree           : 2.40
21 mean indegree            : 2.40
22                               : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
23 symmetric degrees dist. : [0, 0, 0, 0, 1, 4, 0, 0, 0, 0, 0]
24 mean symmetric degree   : 4.80
25 outdegrees concentration index : 0.1667
26 indegrees concentration index : 0.2333
27 symdegrees concentration index : 0.0333
28                               : [0, 1, 2, 3, 4, 'inf']
29 neighbourhood depths distribution: [0, 1, 4, 0, 0, 0]
30 mean neighbourhood depth     : 1.80
31 digraph diameter           : 2
32 agglomeration distribution   :
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

```

These show methods usually rely upon corresponding compute methods, like the `computeSize()`, the `computeDeterminateness()` or the `computeTransitivityDegree()` method (see Listing 1.4 Lines 5,7,16).

```

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

```

Mind that show methods output their results in the *Python* console. We provide also some `showHTML` methods which output their results in a system browser's window.

```

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

```

Relation Map

Ranking rule: Alphabetic

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.

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

1.5 Special digraph instances

Some constructors for universal digraph instances, like the `CompleteDigraph`, the `EmptyDigraph` or the `GridDigraph` constructor, are readily available (see Fig. 1.3).

```

1 >>> from digraphs import GridDigraph
2 >>> grid = GridDigraph(n=5,m=5,hasMedianSplitOrientation=True)
3 >>> grid.exportGraphViz('tutorialGrid')
4 *---- exporting a dot file for GraphViz tools -----
5 Exporting to tutorialGrid.dot
6 dot -Grankdir=BT -Tpng TutorialGrid.dot -o tutorialGrid.png

```

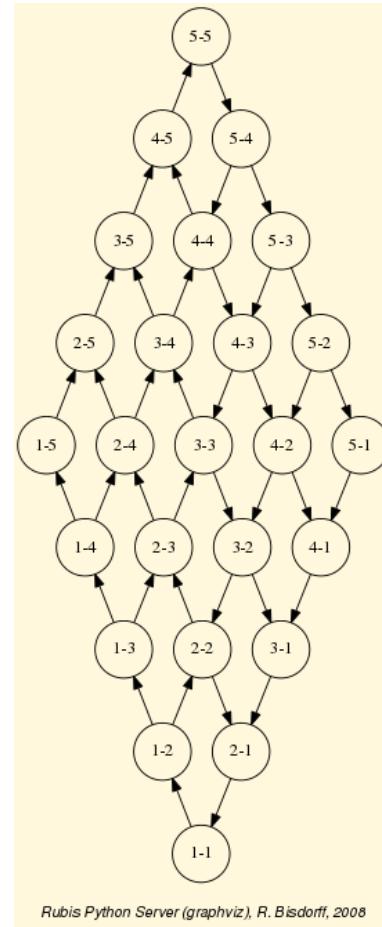


Fig. 1.3 The 5x5 grid graph. Notice the median split orientation. This kind of oriented grids show the highest possible number of chordless circuits.

To be written: chapter conclusion and transition to the next chapter.

Chapter 2

Working with bipolar-valued digraphs

Abstract To be written.

2.1 Random bipolar-valued digraphs

We are starting this tutorial with generating 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 a `randomDigraphs` module providing a specific `RandomValuationDigraph` constructor.

Listing 2.1 Random bipolar-valued digraph instance

```
1  >>> from randomDigraphs import RandomValuationDigraph
2  >>> rdg = RandomValuationDigraph(order=7)
3  >>> rdg.save('tutRandValDigraph')
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 `Digraph.save()` method (see Listing 2.1 Line 3) we may keep a backup version for future use of *rdg* which will be stored in a file called `tutRandValDigraph.py` in the current working directory. The genuine `Digraph` class constructor may restore the *rdg* object from the stored file (Line 4-5). We may easily inspect the content of *rdg* (Line 5). The digraph size 22 indicates the number

of positively valued arcs. The valuation domain is uniformly distributed in the interval $[-1.0; 1.0]$ and the mean absolute arc valuation is $(0.7524 \times 2) - 1.0 = 0.5048$ (Line 12).

All objects of `Digraph` type contain at least the list of attributes shown here: a name (string), a dictionary of actions (digraph nodes), an `order` (integer) attribute containing the number of actions, a `valuationdomain` dictionary, a double dictionary `relation` representing the adjacency table of the digraph relation, a `gamma` and a `notGamma` dictionary containing the direct neighbourhood of each action.

As mentioned in the previous Chapter, the `Digraph` class provides some generic show methods for exploring a given `Digraph` object, like the `showShort()`, `showAll()`, `showRelationTable()` and the `showNeighborhoods()` methods.

Listing 2.2 Example of random valuation digraph

```

1  >>> rdg.showAll()
2  ----- show detail -----
3  Digraph : tutRandValDigraph
4  ----- Actions ----*
5  ['1', '2', '3', '4', '5', '6', '7']
6  ----- Characteristic valuation domain ----*
7  {'med': Decimal('0.0'), 'hasIntegerValuation': False,
8   'min': Decimal('-1.0'), 'max': Decimal('1.0')}
9  ----- Relation Table -----
10 r(xSy) | '1' '2' '3' '4' '5' '6' '7'
11 -----|-----
12 '1' | 0.00 -0.48 0.70 0.86 0.30 0.38 0.44
13 '2' | -0.22 0.00 -0.38 0.50 0.80 -0.54 0.02
14 '3' | -0.42 0.08 0.00 0.70 -0.56 0.84 -1.00
15 '4' | 0.44 -0.40 -0.62 0.00 0.04 0.66 0.76
16 '5' | 0.32 -0.48 -0.46 0.64 0.00 -0.22 -0.52
17 '6' | -0.84 0.00 -0.40 -0.96 -0.18 0.00 -0.22
18 '7' | 0.88 0.72 0.82 0.52 -0.84 0.04 0.00
19 ----- Connected Components ---*
20 1: ['1', '2', '3', '4', '5', '6', '7']
21 Neighborhoods:
22 Gamma:
23 '1': in => {'5', '7', '4'}, out => {'5', '7', '6', '3', '4'}
24 '2': in => {'7', '3'}, out => {'5', '7', '4'}
25 '3': in => {'7', '1'}, out => {'6', '2', '4'}
26 '4': in => {'5', '7', '1', '2', '3'}, out => {'5', '7',
   '1', '6'}
27 '5': in => {'1', '2', '4'}, out => {'1', '4'}
28 '6': in => {'7', '1', '3', '4'}, out => set()
29 '7': in => {'1', '2', '4'}, out => {'1', '2', '3', '4', '6'}
30 Not Gamma:
31 '1': in => {'6', '2', '3'}, out => {'2'}
32 '2': in => {'5', '1', '4'}, out => {'1', '6', '3'}
33 '3': in => {'5', '6', '2', '4'}, out => {'5', '7', '1'}
34 '4': in => {'6'}, out => {'2', '3'}
35 '5': in => {'7', '6', '3'}, out => {'7', '6', '2', '3'}
36 '6': in => {'5', '2'}, out => {'5', '7', '1', '3', '4'}

```

```
37     '7': in => {'5', '6', '3'}, out => {'5'}
```

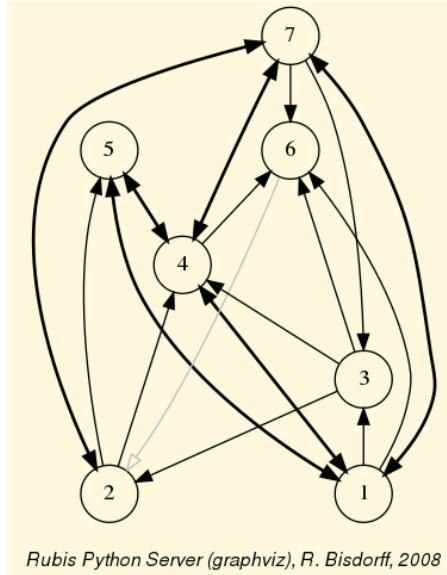
Mind that most Digraph class methods will ignore the *reflexive* links by considering that they are *ineterminate*, i.e. the characteristic value $r(xSx)$ for all action x is set to the *median*, i.e. *ineterminate* value 0.0 in this case (see Listing 2.2 Lines 12-18 and [BIS-2004a]).

2.2 Graphviz drawings

We may have an even better insight into the Digraph object *rdg* by looking at a [graphviz](#) drawing [Footnote 1].

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

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 ($r(6S2) = 0.00$) observed between nodes '6' and '2'. The corresponding link is marked in gray with an open arrowhead in the drawing.



2.3 Asymmetric and symmetric parts

We may now extract both the *symmetric* as well as the asymmetric part of digraph *dg* with the help of two corresponding constructors (see Listing 2.3).

Listing 2.3 Computing asymmetric and symmetric Parts

```

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

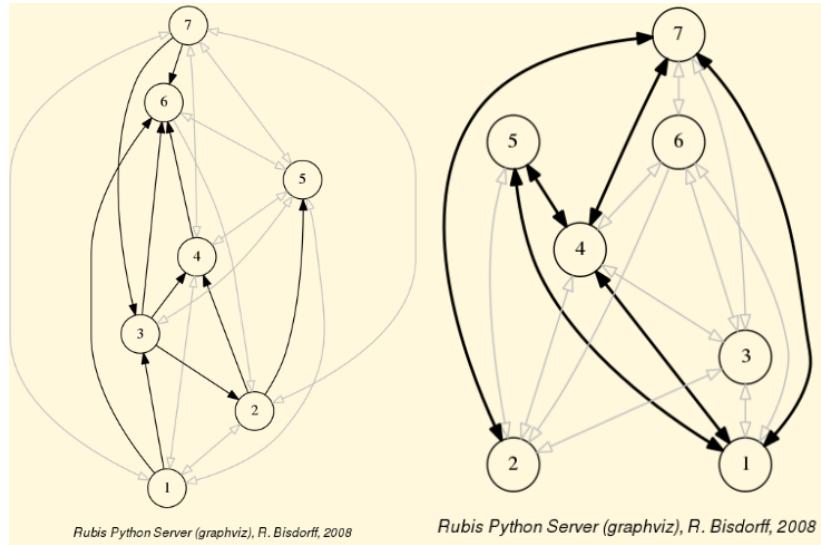


Fig. 2.2 Asymmetric and symmetric part of the tutorial random valuation digraph.

The constructor of the partial objects *asymDg* and *symDg* puts to the indeterminate characteristic value all not-asymmetric, respectively not-symmetric links between nodes (see Fig. 2.2).

Here below, for illustration the source code of the `relation` constructor of the `AsymmetricPartialDigraph` class.

```

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
13                relationIn[b][a] <= Med:
14                    relationOut[a][b] = relationIn[a][b]
15                elif relationIn[a][b] <= Med and
16                    relationIn[b][a] >= Med:
17                        relationOut[a][b] = relationIn[a][b]
18                    else:
19                        relationOut[a][b] = Med
20            else:
21                relationOut[a][b] = Med
22
23    return relationOut

```

2.4 Border and inner parts

We may also extract the border -the part of a digraph induced by the union of its initial and terminal prekernels (see Kernel Chapter)- as well as, the inner part -the complement of the border- with the help of two corresponding class constructors: `GraphBorder` and `GraphInner` (see Fig. 2.3).

Let us illustrate these parts on a linear ordering obtained from the tutorial random valuation digraph `rdg` with the NETFLOWS ranking rule (see Section 8.3).

Listing 2.4 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)
7 >>> bnf.exportGraphViz(worstChoice=['6'], bestChoice=['7'])
8 >>> inf = GraphInner(nf)
9 >>> inf.exportGraphViz(worstChoice=['6'], bestChoice=['7'])

```

We may orient the `graphviz` drawings in Fig. 2.3 with the terminal node 6 (`worstChoice` parameter) and initial node 7 (`bestChoice` parameter) (see Listing 2.4 Lines 7 and 9).

The constructor of the partial digraphs `bnf` and `inf` (see Lines 3 and 6) puts to the *indeterminate* characteristic value all links not in the *border*, respectively *not* in the *inner* part (see Fig. 2.3).

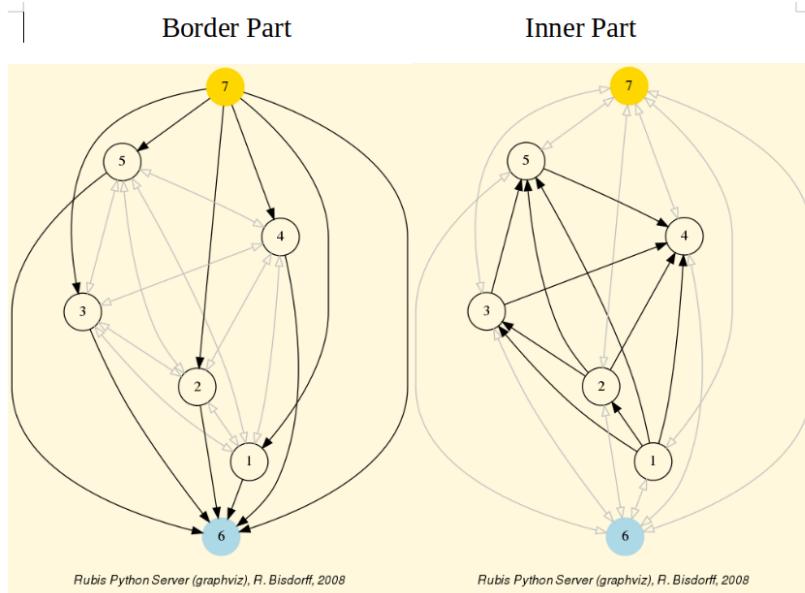


Fig. 2.3 Border and inner part of a linear order oriented by *terminal* and *initial* kernels.

Being much *denser* than a linear order, the actual inner part of our tutorial random valuation digraph dg is reduced to a single arc between nodes 3 and 4 (see Fig. 2.4).

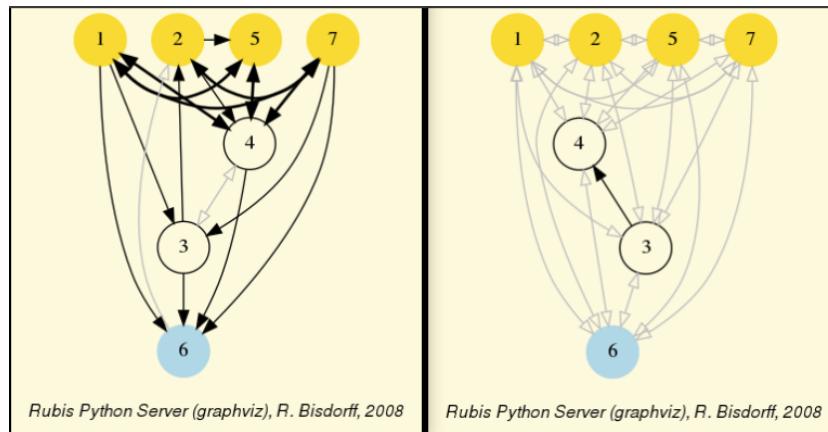


Fig. 2.4 Border and inner part of the tutorial random valuation digraph rdg .

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

We may recover object rdg from both partial objects $asymDg$ and $symDg$, or as well from the border bg and the inner part ig , with a **bipolar fusion** constructor, also called **epistemic disjunction**, available via the `FusionDigraph` class.

Listing 2.5 Epistemic fusion of partial digraphs

```

1  >>> from digraphs import FusionDigraph
2  >>> fusDg = FusionDigraph(asymDg,symDg,operator='o-max')
3  >>> # fusDg = FusionDigraph(bg,ig,operator='o-max')
4  >>> fusDg.showRelationTable()
5  * ---- Relation Table ----
6  r(xSy) | '1'   '2'   '3'   '4'   '5'   '6'   '7'
7  -----|-----
8  '1'    | 0.00 -0.48  0.70  0.86  0.30  0.38  0.44
9  '2'    | -0.22  0.00 -0.38  0.50  0.80 -0.54  0.02
10 '3'    | -0.42  0.08  0.00  0.70 -0.56  0.84 -1.00
11 '4'    | 0.44 -0.40 -0.62  0.00  0.04  0.66  0.76
12 '5'    | 0.32 -0.48 -0.46  0.64  0.00 -0.22 -0.52
13 '6'    | -0.84  0.00 -0.40 -0.96 -0.18  0.00 -0.22
14 '7'    | 0.88  0.72  0.82  0.52 -0.84  0.04  0.00

```

The epistemic fusion operator $o\text{-max}$ (see Listing 2.5 Line 2) works as follows.

Let r and r' characterise two bipolar-valued epistemic situations.

- $o\text{-max}(r, r') = \max(r, r')$ when both r and r' are more or less valid or indeterminate;
- $o\text{-max}(r, r') = \min(r, r')$ when both r and r' are more or less invalid or indeterminate;
- $o\text{-max}(r, r') = 0.0$, i.e. indeterminate otherwise.

2.6 Dual, converse and codual digraphs

We may as readily compute the **dual** (negated relation Footnote[14]), the **converse** (transposed relation) and the **codual** (transposed and negated relation) of the digraph instance rdg .

Listing 2.6 Computing dual, converse and codual

```

1 >>> from digraphs import DualDigraph, ConverseDigraph,
2           CoDualDigraph
3 >>> ddg = DualDigraph(rdg)
4 >>> ddg.showRelationTable()
5  -r(xSy) | '1'   '2'   '3'   '4'   '5'   '6'   '7'
6  -----|-----
7  '1'    | 0.00  0.48 -0.70 -0.86 -0.30 -0.38 -0.44
8  '2'    | 0.22  0.00  0.38 -0.50  0.80  0.54 -0.02
9  '3'    | 0.42  0.08  0.00 -0.70  0.56 -0.84  1.00
10 '4'   | -0.44  0.40  0.62  0.00 -0.04 -0.66 -0.76

```

```

10      '5' | -0.32  0.48  0.46 -0.64  0.00  0.22  0.52
11      '6' |  0.84  0.00  0.40  0.96  0.18  0.00  0.22
12      '7' |  0.88 -0.72 -0.82 -0.52  0.84 -0.04  0.00
13
14 >>> cdg = ConverseDigraph(rdg)
15 >>> cdg.showRelationTable()
16     * ----- Relation Table -----
17     r(ySx) | '1'   '2'   '3'   '4'   '5'   '6'   '7'
18     -----
19     '1' |  0.00 -0.22 -0.42  0.44  0.32 -0.84  0.88
20     '2' | -0.48  0.00  0.08 -0.40 -0.48  0.00  0.72
21     '3' |  0.70 -0.38  0.00 -0.62 -0.46 -0.40  0.82
22     '4' |  0.86  0.50  0.70  0.00  0.64 -0.96  0.52
23     '5' |  0.30  0.80 -0.56  0.04  0.00 -0.18 -0.84
24     '6' |  0.38 -0.54  0.84  0.66 -0.22  0.00  0.04
25     '7' |  0.44  0.02 -1.00  0.76 -0.52 -0.22  0.00
26
27 >>> cddg = CoDualDigraph(rdg)
28 >>> cddg.showRelationTable()
29     * ----- Relation Table -----
30     -r(ySx) | '1'   '2'   '3'   '4'   '5'   '6'   '7'
31     -----
32     '1' |  0.00  0.22  0.42 -0.44 -0.32  0.84 -0.88
33     '2' |  0.48  0.00 -0.08  0.40  0.48  0.00 -0.72
34     '3' | -0.70  0.38  0.00  0.62  0.46  0.40 -0.82
35     '4' | -0.86 -0.50 -0.70  0.00 -0.64  0.96 -0.52
36     '5' | -0.30 -0.80  0.56 -0.04  0.00  0.18  0.84
37     '6' | -0.38  0.54 -0.84 -0.66  0.22  0.00 -0.04
38     '7' | -0.44 -0.02  1.00 -0.76  0.52  0.22  0.00

```

Computing the dual, respectively the converse of a digraph, may also be done with prefixing the `__neg__` (`-`) or the `__invert__` (`~`) operator. The codual of a Digraph object may, hence, as well be computed with a **composition** (in either order) of both operations.

Listing 2.7 Computing the dual, the converse and the codual of a digraph

```

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

```

2.7 Symmetric and transitive closures

Symmetric and transitive closures, by default in-site constructors, are also available (see Fig. 2.5). Note that it is a good idea, before going ahead with these in-site operations, who irreversibly modify the original `rdg` object, to previously make a backup version of `rdg`. The simplest storage method, always provided by the generic `Digraph.save()`, writes out in a named file the python content of the `Digraph` object in string representation (see Section ??).

Listing 2.8 Symmetric and transitive closures

```

1 >>> rdg.save('tutRandValDigraph')
2 >>> rdg.closeSymmetric(InSite=True)
3 >>> rdg.closeTransitive(InSite=True)
4 >>> rdg.exportGraphViz('strongComponents')

```

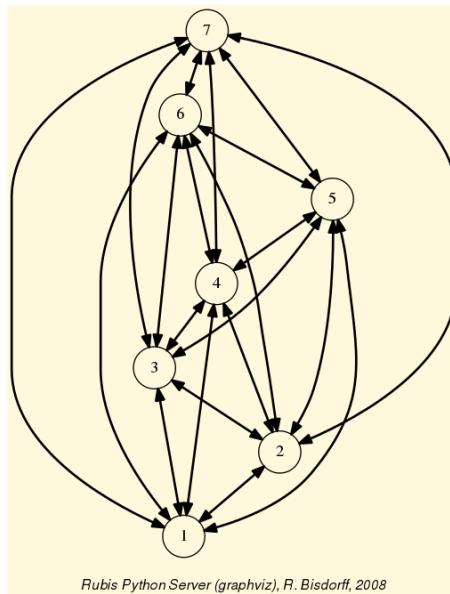


Fig. 2.5 Symmetric and transitive closure of the tutorial random valuation digraph `rdg`.

The `closeSymmetric()` method (see Listing 2.8 Line 2), of complexity $O(n^2)$ where n denotes the digraph's order, changes, on the one hand, all single pairwise links it may detect into double links by operating a disjunction of the pairwise relations. On the other hand, the `closeTransitive()` method (see Line 3), implements the *Roy-Warshall* transitive closure algorithm of complexity $O(n^3)$. (Footnote [17])

The same closeTransitive() method with a Reverse = True flag may be readily used for eliminating all transitive arcs from a transitive digraph instance. We make usage of this feature when drawing Hasse diagrams of TransitiveDigraph objects.

2.8 Strong components

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

Listing 2.9 Computing strong components

```

1 >>> from digraphs import StrongComponentsCollapsedDigraph
2 >>> sc = StrongComponentsCollapsedDigraph(dg)
3 >>> sc.showAll()
4     ----- show detail -----
5     Digraph          : tutRandValDigraph_Scc
6
7     ----- Actions -----
8     ['_7_1_2_6_5_3_4_']
9
10    * ---- Relation Table ----
11      S      | 'Scc_1'
12      -----|-----
13
14      'Scc_1' | 0.00
15      short      content
16      Scc_1      _7_1_2_6_5_3_4_
17
18      Neighborhoods:
19      Gamma      :
20      'frozenset({'7','1','2','6','5','3','4'})':
21                  in => set(), out => set()
22      Not Gamma :
23      'frozenset({'7','1','2','6','5','3','4'})':
24                  in => set(), out => set()
```

2.9 CSV storage

Sometimes it is required to exchange the graph valuation data in CSV format with a statistical package like R. For this purpose it is possible to export the digraph data into a CSV file. The valuation domain is hereby normalized by default to the range $[-1.0, 1.0]$ and the diagonal is put by default to the minimal value -1.0 .

```

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 content.

```

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

```

It is as well possible to show a colored version of the valued relation table in a system browser window tab (see Fig. 2.5).

```

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

```

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- links, like the reflexive diagonal ones or the link between node *6* and node *2*, are shown in *gray*.

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	-	0.70	-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

2.10 Complete, empty and indeterminate digraphs

Let us finally mention some special universal classes of digraphs that are readily available in the :py:mod:`digraphs` module, like the `CompleteDigraph`, the `EmptyDigraph` and the `IndeterminateDigraph` classes, which put all characteristic values respectively to the maximum, the minimum or the median indeterminate characteristic value.

Listing 2.10 Complete, empty and indeterminate digraphs

```

1 >>> from digraphs import CompleteDigraph, EmptyDigraph, \
2 ...                           IndeterminateDigraph
3
4 >>> e = EmptyDigraph(order=5)
5 >>> e.showRelationTable()
6     * ---- Relation Table ----
7     S | '1'   '2'   '3'   '4'   '5'
8     ---|-----
9     '1' | -1.00 -1.00 -1.00 -1.00 -1.00
10    '2' | -1.00 -1.00 -1.00 -1.00 -1.00
11    '3' | -1.00 -1.00 -1.00 -1.00 -1.00
12    '4' | -1.00 -1.00 -1.00 -1.00 -1.00
13    '5' | -1.00 -1.00 -1.00 -1.00 -1.00
14
15 >>> e.showNeighborhoods()
16 Neighborhoods:
17     Gamma :
18     '1': in => set(), out => set()
19     '2': in => set(), out => set()
20     '5': in => set(), out => set()
21     '3': in => set(), out => set()
22     '4': in => set(), out => set()
23     Not Gamma :
24     '1': in => {'2', '4', '5', '3'}, out => {'2', '4', '5', '3'}
25     '2': in => {'1', '4', '5', '3'}, out => {'1', '4', '5', '3'}
26     '5': in => {'1', '2', '4', '3'}, out => {'1', '2', '4', '3'}
27     '3': in => {'1', '2', '4', '5'}, out => {'1', '2', '4', '5'}
28     '4': in => {'1', '2', '5', '3'}, out => {'1', '2', '5', '3'}
29
30 >>> i = IndeterminateDigraph()
31     * ---- Relation Table ----
32     S | '1'   '2'   '3'   '4'   '5'
33     ---|-----
34     '1' | 0.00 0.00 0.00 0.00 0.00
35     '2' | 0.00 0.00 0.00 0.00 0.00
36     '3' | 0.00 0.00 0.00 0.00 0.00
37     '4' | 0.00 0.00 0.00 0.00 0.00
38     '5' | 0.00 0.00 0.00 0.00 0.00
39
40 >>> i.showNeighborhoods()
41 Neighborhoods:
42     Gamma :
43     '1': in => set(), out => set()

```

```
44 '2': in => set(), out => set()
45 '5': in => set(), out => set()
46 '3': in => set(), out => set()
47 '4': in => set(), out => set()
48     Not Gamma :
49 '1': in => set(), out => set()
50 '2': in => set(), out => set()
51 '5': in => set(), out => set()
52 '3': in => set(), out => set()
53 '4': in => set(), out => set()
```

Mind the subtle difference between the neighborhoods of an **empty** and the neighborhoods of an **indeterminate** digraph instance. In the first kind, the neighborhoods are known to be completely *empty* (see Listing 2.10 Lines 22-27) whereas, in the latter, *nothing is known* about the actual neighborhoods of the nodes (see Lines 45-50). These two cases illustrate why in the case of **bipolar-valued** digraphs, we may need both a `gamma` **and** a `notGamma` attribute.

Chapter 3

Working with outranking digraphs

Abstract To be written.

3.1 Outranking digraph model

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 `OutrankingDigraph` type. A given object contains hence the following attributes:

1. A ordered dictionary of decision **actions** describing the potential decision actions or alternatives with name and comment attributes,
2. A possibly empty ordered dictionary of decision **objectives** with name and comment attributes, describing the multiple preference dimensions involved in the decision problem,
3. An ordered dictionary of performance **criteria**, i.e. *preferentially independent* and *non-redundant* decimal-valued functions used for measuring the performance of each potential decision action with respect to a decision objective,
4. A double dictionary called **evaluation** gathering performance grades for each decision action or alternative on each criterion function.
5. The digraph **valuationdomain**, a dictionary with three entries: the *minimum* (-1.0, certainly outranked), the *median* (0.0, indeterminate) and the *maximum* (+1.0, certainly outranking),
6. The outranking **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 construct, for instance, a random bipolar-valued outranking digraph with seven decision actions denoted a_1, a_2, \dots, a_7 . We need therefore to first generate a corresponding random performance tableaux (see Listing 3.1 below).

Listing 3.1 Generating a random performance tableau

```

1 >>> from outrankingDigraphs import *
2 >>> pt = RandomPerformanceTableau(numberOfActions=7, \
3 ...                                     seed=100)
4 >>> pt
5       *----- PerformanceTableau instance description -----*
6       Instance class      : RandomPerformanceTableau
7       Seed                 : 100
8       Instance name       : randomperftab
9       Actions              : 7
10      Criteria             : 7
11      NaN proportion (%) : 6.1
12 >>> pt.showActions()
13       *----- show digraphs actions -----*
14       key: a1
15           name:      action #1
16           comment:   RandomPerformanceTableau() generated.
17       key: a2
18           name:      action #2
19           comment:   RandomPerformanceTableau() generated.
20       ...
21       ...
22       key: a7
23           name:      action #7
24           comment:   RandomPerformanceTableau() generated.
```

In this example we consider furthermore a family of seven **equisignificant 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 grading procedure's 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       *---- criteria ----*
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 $g1$ (see Lines Listing 3.2 6-8 above) we observe, for instance, about 5% of *indifference*, about 90% of *preference* and about 5% of *considerable* performance difference situations. The individual performance evaluation of all decision alternative on each criterion are gathered in a **performance tableau**.

Listing 3.3 Inspecting the performance table

```

1 >>> pt.showPerformanceTableau()
2     *---- performance tableau ----*
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
9     'g5'    | 47.7   14.8   79.7   67.5   NA      90.7   80.2
10    'g6'   | 69.6   45.5   22.0   33.8   31.8   NA      48.8
11    'g7'   | 82.9   41.7   12.8   21.9   75.7   15.4   6.0

```

It is noteworthy to mention the three **missing data** (NA) cases: action $a3$ is missing, for instance, a grade on criterion $g2$ (see Listing 3.3 Line 6 above).

3.2 The bipolar-valued outranking digraph

Given the previous random performance tableau pt , the `BipolarOutrankingDigraph` constructor computes the corresponding **bipolar-valued outranking digraph**.

Listing 3.4 Example of random bipolar-valued outranking digraph

```

1 >>> g = BipolarOutrankingDigraph(pt)
2 >>> g
3     *----- Object instance description -----*
4     Instance class      : BipolarOutrankingDigraph
5     Instance name       : rel_randomperftab
6     Actions             : 7
7     Criteria            : 7
8     Size                : 20
9     Determinateness (%) : 63.27
10    Valuation domain    : [-1.00;1.00]
11    Attributes          : [
12        'name', 'actions',
13        'criteria', 'evaluation', 'NA',
14        'valuationdomain', 'relation',

```

```

15     'order', 'gamma', 'notGamma', ...
16 ]

```

The resulting digraph contains 20 positive (valid) outranking realtions. And, the mean majority criteria significance support of all the pairwise outranking situations is 63.3% (see Listing 3.4 Lines 8-9). We may inspect the complete $[-1.0, +1.0]$ -valued adjacency table as follows.

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

```

Considering the given performance tableau pt , the `BipolarOutrankingDigraph` class constructor computes the characteristic value $r(x,y)$ of a *pairwise outranking* relation $x \succsim y$ (see [BIS-2013], [ADT-L7]) in a default *normalised valuation domain* $[-1.0, +1.0]$ with the *median* value 0.0 acting as **indeterminate** characteristic value. The semantics of $r(x,y)$ are the following.

1. When $r(x,y) > 0.0$, it is more *True* than *False* that x **outranks** y , i.e. alternative x is at least as well performing than alternative y on a weighted majority of criteria **and** there is no considerable negative performance difference observed in disfavour of x ,
2. When $r(x,y) < 0.0$, it is more *False* than *True* that x **outranks** y , i.e. alternative x is **not** at least as well performing on a weighted majority of criteria than alternative y **and** there is no considerable positive performance difference observed in favour of x ,
3. When $r(x,y) = 0.0$, it is **indeterminate** whether x outranks y or not.

3.3 Pairwise comparisons

From above given semantics, we may consider (see Listing 3.5 Line 5 above) that $a1$ outranks $a2$ ($r(a1, a2) > 0.0$), but not $a7$ ($r(a1, a7) = 0.0$). In order to comprehend the characteristic values shown in the relation table above, we may furthermore inspect the details of the pairwise multiple criteria comparison between alternatives $a1$ and $a2$.

Listing 3.6 Inspecting a pairwise multiple criteria comparison

```

1 >>> odg.showPairwiseComparison('a1','a2')

```

```

2   *----- pairwise comparison -----*
3   Comparing actions : (a1, a2)
4   crit. wght. g(x) g(y) diff | ind pref r()
5   -----
6     g1  1.00 15.17 44.51 -29.34 | 2.50 5.00 -1.00
7     g2  1.00 82.29 43.90 +38.39 | 2.50 5.00 +1.00
8     g3  1.00 44.23 19.10 +25.13 | 2.50 5.00 +1.00
9     g4  1.00 46.37 16.22 +30.15 | 2.50 5.00 +1.00
10    g5  1.00 47.67 14.81 +32.86 | 2.50 5.00 +1.00
11    g6  1.00 69.62 45.49 +24.13 | 2.50 5.00 +1.00
12    g7  1.00 82.88 41.66 +41.22 | 2.50 5.00 +1.00
13   -----
14   Valuation in range: -7.00 to +7.00; r(x,y): +5/7 = +0.71

```

The outranking characteristic value $r(a1 \succsim a2)$ represents the **majority margin** resulting from the difference between the weights of the criteria in favor and the weights of the criteria in disfavor of the statement that alternative $a1$ is at least as well performing as alternative $a2$. No considerable performance difference being observed above, no veto or counter-veto situation is triggered in this pairwise comparison. Such a situation is, however, observed for instance when we pairwise compare the performances of alternatives $a1$ and $a7$.

```

1 >>> odg.showPairwiseComparison('a1','a7')
2   *----- pairwise comparison -----*
3   Comparing actions : (a1, a7)
4   crit. wght. g(x) g(y) diff | ind pref r() | v
5   veto
6   -----
7     g1  1.00 15.17 96.58 -81.41 | 2.50 5.00 -1.00 | 80.00
8     -1.00
9     g2  1.00 82.29 62.22 +20.07 | 2.50 5.00 +1.00 |
10    g3  1.00 44.23 56.90 -12.67 | 2.50 5.00 -1.00 |
11    g4  1.00 46.37 32.06 +14.31 | 2.50 5.00 +1.00 |
12    g5  1.00 47.67 80.16 -32.49 | 2.50 5.00 -1.00 |
13    g6  1.00 69.62 48.80 +20.82 | 2.50 5.00 +1.00 |
14    g7  1.00 82.88 6.05 +76.83 | 2.50 5.00 +1.00 |
15   -----
16   Valuation in range: -7.00 to +7.00; r(x,y)= +1/7 => 0.0

```

This time, we observe a 57.1% majority of criteria significance $[(1/7+1)/2 = 0.571]$ warranting an *as well as performing* situation. Yet, we also observe a considerable negative performance difference on criterion $g1$ (see Listing 3.5 first row in the relation table above). Both contradictory facts trigger eventually an *indeterminate* outranking situation [BIS-2013].

3.4 Recoding the digraph valuation

All outranking digraphs, being of root type `Digraph`, inherit the methods available under this latter class. The characteristic valuation domain of a digraph may, for

instance, be recoded with the `recodeValutaion()` method below to the *integer* range $[-7, +7]$, i.e. plus or minus the global significance of the family of criteria considered in this example instance.

Listing 3.7 Recoding the digraph valuation

```

1 >>> odg.recodeValuation(-37,+37)
2 >>> odg.valuationdomain['hasIntegerValuation'] = True
3 >>> Digraph.showRelationTable(odg,ReflexiveTerms=False)
4   * ----- Relation Table -----
5   r(x,y) | 'a1' 'a2' 'a3' 'a4' 'a5' 'a6' 'a7'
6   -----|-----
7   'a1' | 0 5 2 2 2 2 0
8   'a2' | -5 0 -1 -1 1 2 -4
9   'a3' | -1 2 0 -1 -1 0 -1
10  'a4' | 0 1 4 0 2 4 -3
11  'a5' | -1 0 1 0 0 2 -1
12  'a6' | -1 0 1 -1 1 0 0
13  'a7' | 0 5 4 3 2 0 0
14 Valuation domain: [-7;+7]

```

Notice that the reflexive self comparison characteristic $r(x,x)$ is set above by default to the median indeterminate valuation value 0; the reflexive terms of binary relation being generally ignored in most of the DIGRAPH3 resources.

3.5 The strict outranking digraph

From the theory (see [BIS-2013], [ADT-L7]) we know that a bipolar-valued outranking digraph is **weakly complete**, i.e. if $r(x,y) < 0.0$ then $r(y,x) \geq 0.0$. From this property follows that a bipolar-valued outranking relation verifies the **coduality** principle: the **dual** (*strict negation* – Footnote[14]) of the **converse** (*inverse* \approx) of the outranking relation corresponds to its *strict outranking* part.

We may visualize the **codual** (*strict*) outranking digraph with a graphviz drawing Footnote[1].

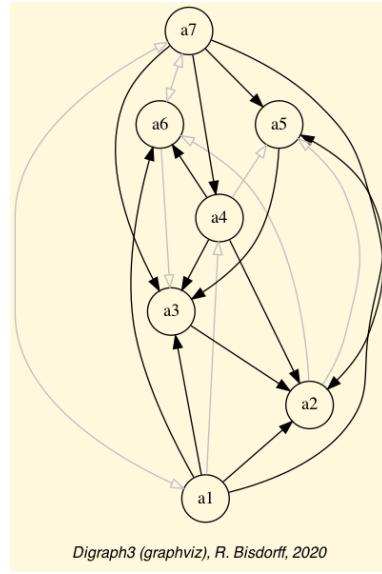
```

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

```

Many more tools for exploiting bipolar-valued outranking digraphs are available in the DIGRAPH3 resources (see the technical documentation of the `outrankingDigraphs` module and the `perfTabs` module).

Fig. 3.1 The codual outranking digraph. It becomes readily clear now from the picture above 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 winner* and may be recommended as potential *best decision actions* in this illustrative preference modelling exercise.



Part II

**Evaluation models and decision methods
and tools**

To be written ...

Chapter 4

How to create a new performance tableau instance

Abstract To be written.

In this tutorial we illustrate a way of creating a new *PerformanceTableau* instance by editing a template with 5 decision alternatives, 3 decision objectives and 6 performance criteria.

4.1 Editing a template file

For this purpose we provide the following *perfTab_Template.py* file in the *examples* directory of the DIGRAPH3 resources.

Listing 4.1 PerformanceTableau object template

```
1 #####  
2 # Digraph3 documentation  
3 # Template for creating a new PerformanceTableau instance  
4 # (C) R. Bisдорff 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', {  
15         'shortName': 'action1',  
16         'name': 'decision alternative a1',  
17         'comment': 'some specific features of this alternative',  
18         }),  
19         ...  
20         ...  
21     ])  
#####
```

```

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

```

```

77     'a3': Decimal("6"),
78     'a4': Decimal('2'),
79     'a5': Decimal('9'),
80   },
81   # g3 has preferenceDirection = 'min'
82   'g3': {
83     'a1': Decimal("-52.2"),
84     'a2': NA,
85     'a3': Decimal("-47.3"),
86     'a4': Decimal('-35.7'),
87     'a5': Decimal('-68.00'),
88   },
89   ...
90   ...
91 }
#####

```

The template file, shown in Listing 4.1, contains first the instructions to import the required `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 **evaluation**.

4.2 Editing the decision alternatives

Decision alternatives are stored in attribute **actions** under the `OrderedDict` format (see the [OrderedDict](#) description in the Python documentation).

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

Mind that graphviz drawings require digraph actions' (nodes) identifier strings without any special characters like '`_`' or '`/`'.

Decision actions descriptions are stored in the order of which they appear in the stored instance file. The `OrderedDict` object keeps this given order when iterating over the decision alternatives.

The random performance tableau models presented in the previous tutorial 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. The *3-Objectives* model keeps a detailed record of the performance profile per decision objective and the corresponding random generators per performance criteria (see Lines 7- below).

```

1 >>> t = Random3ObjectivesPerformanceTableau()
2 >>> t.actions

```

```

3     OrderedDict([
4         ('p01', {'shortName': 'p01',
5             'name': 'action_p01_Eco~Soc~Env+', 
6             'comment': 'random public policy',
7             'Eco': 'fair',
8             'Soc': 'weak',
9             'Env': 'good',
10            'profile': {'Eco': 'fair',
11                'Soc': 'weak',
12                'Env': 'good'}
13            'generators': {'ec01': ('triangular', 50.0, 0.5),
14                'so02': ('triangular', 30.0, 0.5),
15                'en03': ('triangular', 70.0, 0.5),
16                ...
17            },
18        ),
19        ...
20    ],
21)

```

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

4.3 Editing the decision objectives

The minimal required attributes (see Listing 4.1 Lines 27-33) of the ordered decision **objectives** dictionary, besides the individual objective identifiers, are `name`, `comment`, `criteria` (the list of significant performance criteria) and `weight` (the importance of the decision objective). The latter attribute contains the sum of the *significance* weights of the objective's criteria list.

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

Listing 4.2 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')))
20 ]
)

```

The importance weight sums up to 48 for each one of the three example decision objectives shown in Listing 4.2 (Lines 8,13 and 19), so that the significance of each one of the 3 economic criteria is set to 16, of both societal criteria is set to 24, and of each one of the 6 environmental criteria is set to 8.

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 in fact its own decision objective. The criterion significance equals in this case the corresponding decision objective's importance weight.

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

4.4 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 grading functions gathered in an ordered `criteria` dictionary. The required attributes (see Listing 4.3), besides the criteria identifiers, are the usual `shortName`, `name` and `comment`. Specific for a criterion are furthermore the `objective` reference, the significance `weight`, the grading `scale` (minimum and maximum performance values), the `preferenceDirection` ('max' or 'min') and the `performance thresholds` attributes.

Listing 4.3 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     ... ]
)
```

In our bipolar-valued outranking approach, all performance criteria implement *decimal-valued* grading functions, where preferences are either *increasing* or *decreasing* with measured performances.

In order to model a **coherent** performance tableau, the decision criteria must satisfy two methodological requirements:

1. **Independance**: Each decision criterion implements a grading that is *functionally independent* of the grading of the other decision criteria, i.e. the performance measured on one of the criteria does not *constrain* the performance measured on any other criterion.
2. **Non redundancy**: Each performance criterion is only *significant* for a *single* decision objective.

In order to take into account any, usually *unavoidable*, **imprecision** of the performance grading procedures, we may specify three performance **discrimination thresholds**: an *indifference* ('ind'), a *preference* ('pref') and a *considerable performance difference* ('veto') threshold (see Listing 4.3 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 g_1 , shown in Listing 4.3, we observe for instance a grading 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 [BIS-2013].

In a random *Cost-Benefit* performance tableau model we may obtain by default the following content.

Listing 4.4 Example of cardinal Costs criterion

```

1 >>> tcb = RandomCBPerformanceTableau()
2 >>> tcb.showObjectives()
3 *----- decision objectives -----*
4 C: Costs
5   c1 random cardinal cost criterion 6
6   Total weight: 6.00 (1 criteria)
7   ...
8   ...
9 >>> tcb.criteria
10 OrderedDict([
11   ('c1', {'preferenceDirection': 'min',
12           'scaleType': 'cardinal',
13           'objective': 'C',
14           'shortName': 'c1',
15           'name': 'random cardinal cost criterion',
```

```

16     'scale': (0.0, 100.0),
17     'weight': Decimal('6'),
18     'randomMode': ['triangular', 50.0, 0.5],
19     'comment': 'Evaluation generator: triangular law ...',
20     'thresholds': OrderedDict([
21         ('ind', (Decimal('1.49'), Decimal('0'))),
22         ('pref', (Decimal('3.7'), Decimal('0'))),
23         ('veto', (Decimal('67.71'), Decimal('0'))),
24     ]),
25 },
26 ...
27 ...
28 ])

```

Criterion c_1 appears here (see Listing 4.4) to be a cardinal criterion to be minimized and significant for the *Costs* (C) decision objective. We may use the `showCriteria()` method for printing the corresponding performance discrimination thresholds.

```

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

```

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 a *polarisation* of the corresponding outranking statements.

After the performance criteria description, we are ready for recording the actual **performance table**.

4.5 Editing the performance table

The individual grades of each decision alternative on each decision criterion are recorded in a double *criterion x action* dictionary called `evaluation` (see Listing 4.4). As we may encounter missing data cases, we previously define a *missing data* symbol `NA` which is set here to a value disjoint from all the measurement scales, by default `Decimal('−999')` (Line 2).

Listing 4.5 Editing performance grades

```

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 4.4 (Lines 16-) that on a criterion with `preferenceDirection = 'min'` all performance grades are recorded as **negative** values.

We may now inspect the eventually recorded complete template performance table.

```

1 >>> from perfTabs import PerformanceTableau
2 >>> t = PerformanceTableau('perfTab_Template')
3 >>> t.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

```

We may furthermore compute the associated outranking digraph and check if we observe any polarised outranking situations.

```

1 >>> from outrankingDigraphs import BipolarOutrankingDigraph
2 >>> g = BipolarOutrankingDigraph(t)
3 >>> g.showVetos()
4 ----- Veto situations ---
5 number of veto situations : 1
6 1: r(a4 >= a2) = -0.44
7   criterion: g1
8   Considerable performance difference : -77.00
9   Veto discrimination threshold       : -60.00
10  Polarisation: r(a4 >= a2) = -0.44 ==> -1.00
11 ----- Counter-veto situations ---

```

```

12   number of counter-veto situations : 1
13   1: r(a2 >= a4) = 0.56
14     criterion: g1
15     Considerable performance difference : 77.00
16     Counter-veto threshold : 60.00
17     Polarisation: r(a2 >= a4) = 0.56 ==> +1.00

```

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

4.6 Inspecting the template outranking relation

Let us have a look at the outranking relation table.

Listing 4.6 The template outranking relation

```

1 >>> g.showRelationTable()
2 * ----- Relation Table -----
3   r | 'a1' 'a2' 'a3' 'a4' 'a5'
4   -----|-----
5   'a1' | +1.00 -0.44 -0.22 -0.11 +0.06
6   'a2' | +0.44 +1.00 +0.33 +1.00 +0.28
7   'a3' | +0.67 -0.33 +1.00 +0.00 +0.17
8   'a4' | +0.11 -1.00 +0.00 +1.00 +0.06
9   'a5' | -0.06 -0.06 -0.17 -0.06 +1.00

```

We may notice in the outranking relation table above (see Listing 4.6) 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 may orient this way the *graphviz* drawing of the template outranking digraph.

```

1 >>> g.exportGraphViz(fileName= 'template', \
2 ...                         bestChoice =[ 'a2' ], \
3 ...                         worstChoice=[ 'a5' ])
4 *----- exporting a dot file for GraphViz tools -----
5   Exporting to template.dot
6   dot -Grankdir=BT -Tpng template.dot -o template.png

```

In Listing 4.6 their pairwise outranking characteristics show indeed the **indeterminate** value 0.00 (Lines 7-8). We may check their pairwise comparison as follows.

```

1 >>> g.showPairwiseComparison('a3', 'a4')
2 *----- pairwise comparison -----
3   Comparing actions : ('a3', 'a4')
4   crit. wght. g(x) g(y) diff | ind pref r() |
5   -----|-----
6   'g1' 3.00 63.00 23.00 +40.00 | 2.50 5.00 +3.00 |
7   'g2' 3.00 6.00 2.00 +4.00 | 0.00 1.00 +3.00 |
8   'g3' 6.00 -47.30 -35.70 -11.60 | 0.00 10.00 -6.00 |
9   'g4' 2.00 55.40 83.50 -28.10 | 2.09 4.18 -2.00 |

```

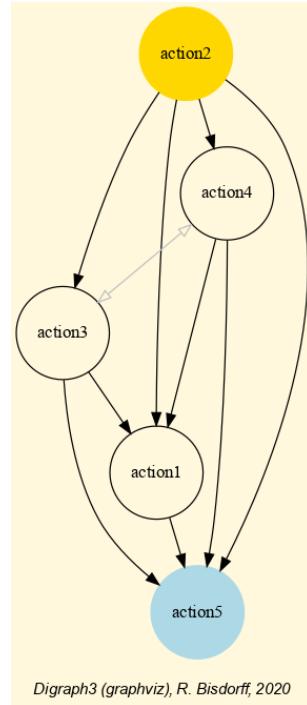


Fig. 4.1 The template outranking digraph models in fact a **partial order** on the five potential decision alternatives. Alternatives *action3* (a_3) and *action4* (a_4) appear actually **incomparable**.

```

10   'g5'  2.00  63.50  37.50  +26.00 |  0.00 10.00  +2.00 |
11   'g6'  NA    54.90
12   Outranking characteristic value: r(a3 >= a4) = +0.00
13   Valuation in range: -18.00 to +18.00

```

The incomparability situation between a_3 and a_4 results here from a perfect balancing of positive (+8) and negative (-8) criteria significances.

4.7 Ranking the template peformance tableau

We may eventually rank the five decision alternatives with a heatmap browser view following the *Copeland* ranking rule (see Section 7.2) which consistently reproduces the partial outranking order shown in Fig. 4.1.

```

1  >>> g.showHTMLPerformanceHeatmap(ndigits=1,colorLevels=5,\n2      ...     Correlations=True,rankingRule='Copeland',\n3      ...     pageTitle='Heatmap of the template performance\n        tableau')

```

Due to a 11 against 7 **plurality tyranny** effect, the *Copeland* ranking rule, essentially based on crisp majority outranking counts, puts here alternative *action5* (a_5)

Heatmap of the template performance tableau

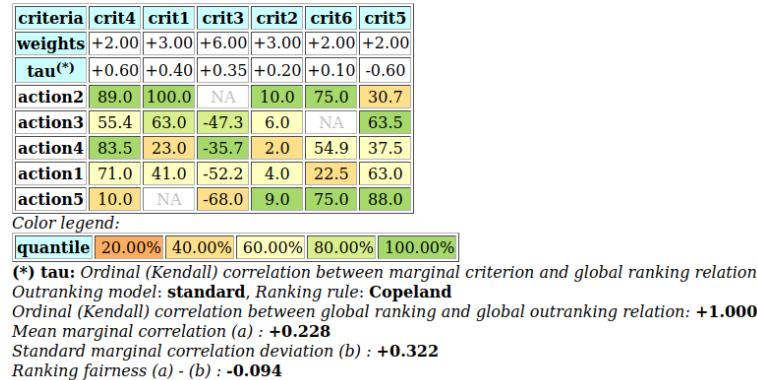


Fig. 4.2 Copeland ranked heatmap of the template performance tableau

last, despite its excellent grades observed on criteria g_2 , g_5 and g_6 . A slightly **fairer** ranking result may be obtained with the *NetFlows* ranking rule.

```

1      >>> g.showHTMLPerformanceHeatmap(ndigits=1,colorLevels=5,\n2          ...     Correlations=True,rankingRule='NetFlows',\n3          ...     pageTitle='Heatmap of the template performance\n        tableau')

```

Heatmap of the template performance tableau

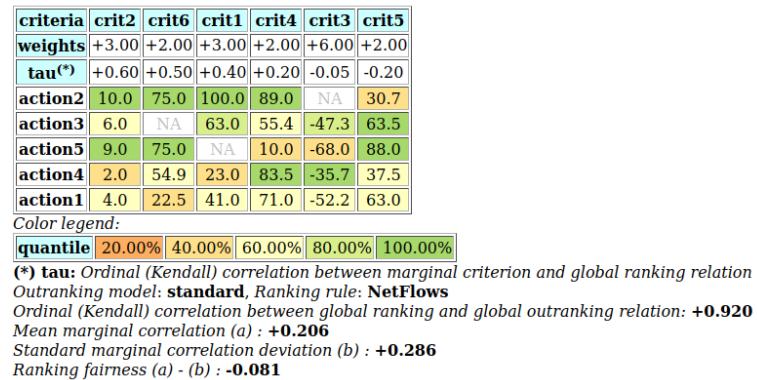


Fig. 4.3 Net flows ranked heatmap of the template performance tableau

It might be opportun to furthermore study the robustness of the apparent outranking situations when assuming only *ordinal* or *uncertain* criteria significance weights. If interested in mainly objectively *unopposed* (multipartisan) outranking

situations, one might also try the `UnOpposedOutrankingDigraph` constructor. (see Chapter ??).

Chapter 5

Generating random performance tableaux

Abstract To be written.

5.1 Introduction

The `randomPerfTabs` module provides several constructors 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 at the University of Luxembourg. This tutorial concerns the most useful models.

The simplest model, called **RandomPerformanceTableau**, generates a set of n decision actions, a set 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 performances are Beta(2,2) distributed on each measurement scale.

One of the most useful models, called **RandomCBPerformanceTableau**, proposes a performance tableau involving two decision objectives, named *Costs* (to be minimized) respectively *Benefits* (to be maximized); its purpose being to generate more or less contradictory performances on these two, usually conflicting, objectives. *Low costs* will randomly be coupled with *low benefits*, whereas *high costs* will randomly be coupled with *high benefits*.

Many public policy decision problems involve three often conflicting decision objectives taking into account *economical*, *societal* as well as *environmental* aspects. For this type of performance tableau model, we provide a specific model, called **Random3ObjectivesPerformanceTableau**.

Deciding which students, based on the grades obtained in a number of examinations, validate or not their academic studies, is the genuine decision practice of universities and academies. To thoroughly study these kind of decision problems, we provide a corresponding performance tableau model, called **RandomAcademicPer-**

formanceTableau, which gathers grades obtained by a given number of students in a given number of weighted courses.

In order to study aggregation of election results (see Chapter 7) in the context of bipolar-valued outranking digraphs, we provide furthermore a specific performance tableau model called **RandomRankPerformanceTableau** which provides ranks (linearly ordered performances without ties) of a given number of election candidates (decision actions) for a given number of weighted voters (performance criteria).

5.2 Random standard performance tableaux

The RandomPerformanceTableau class, the simplest of the kind, specializes the generic PerformanceTableau class, and takes the following parameters:

- `numberOfActions := nbr` of decision actions.
- `numberOfCriteria := number` performance criteria.
- `weightDistribution := 'random'` (default) — `'fixed'` — `'equisignificant'`:
 - If `'random'`, weights are uniformly selected randomly from the given weight scale;
 - If `'fixed'`, the `weightScale` must provided a corresponding weights distribution;
 - If `'equisignificant'`, all criterion weights are put to unity.
- `weightScale := [Min,Max]` (default =`(1,numberOfCriteria)`).
- `IntegerWeights := True` (default) — `False` (normalized to proportions of 1.0).
- `commonScale := [a,b];` common performance measuring scales (default = `[0.0,100.0]`)
- `commonThresholds := [(q0,q1),(p0,p1),(v0,v1)]`; common indifference(q), preference (p) and considerable performance difference (v) discrimination thresholds. For each threshold type $x \in \{q, p, v\}$, the float $x0$ value represents a constant percentage of the common scale and the float $x1$ value a proportional value of the actual performance measure. Default values are `[(2.5,0,0.0),(5.0,0,0),(60.0,0,0)]`.
- `commonMode := common random distribution of random performance measurements` (default = `('beta',None,(2,2))`):
 - `('uniform', None, None)`, uniformly distributed float values on the given common scales' range `[Min,Max]`;
 - `('normal', μ , σ)`, truncated Gaussian distribution, by default $\mu = (b - a)/2$ and $\sigma = (b - a)/4$;
 - `('triangular', mode, repartition)`, generalized triangular distribution with a probability repartition parameter specifying the probability mass accumulated until the mode value. By default, `mode = (b - a)/2` and `*repartition* = 0.5`.

- ('beta',None,(alpha,beta)), a beta generator with default alpha=2 and beta=2 parameters.
- valueDigits := integer, precision of performance measurements (2 decimal digits by default).
- missingDataProbability := $0.0 \leq \text{float} \leq 1.0$; probability of missing performance evaluation on a criterion for an alternative (default 0.025).
- NA := Decimal (default = -999); missing data symbol.

Code example:

Listing 5.1 Generating a random performance tableau

```

1 >>> from randomPerfTabs import RandomPerformanceTableau
2 >>> t = RandomPerformanceTableau(numberOfActions=21,\ 
3 ...                               numberOfCriteria=13, seed=100)
4 >>> t.actions
5 { 'a01': { 'comment': 'RandomPerformanceTableau() generated.' ,
6       'name': 'random decision action' },
7     'a02': { ... },
8 ...
9   }
10 >>> t.criteria
11 { 'g01': { 'thresholds': {
12       'ind': (Decimal('10.0'), Decimal('0.0')),
13       'veto': (Decimal('80.0'), Decimal('0.0')),
14       'pref': (Decimal('20.0'), Decimal('0.0'))},
15       'scale': [0.0, 100.0],
16       'weight': Decimal('1'),
17       'name': 'digraphs.RandomPerformanceTableau() instance',
18       'comment': 'Arguments: ; weightDistribution=random;
19           weightScale=(1, 1); commonMode=None'},
20       'g02': { ... },
21     ...
22   }
23 >>> t.evaluation
24 { 'g01': { 'a01': Decimal('15.17') ,
25       'a02': Decimal('44.51') ,
26       'a03': Decimal('-999'), # missing evaluation
27       ... },
28   ...
29 >>> t.showHTMLPerformanceTableau()

```

Missing (NA) evaluation are registered in a performance tableau by default as `Decimal(' -999')` value (see Listing 5.1 Line 24). Best and worst performance on each criterion are marked in *light green*, respectively in *light red*.

Performance table randomperftab

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	NA	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	NA	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	NA	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	NA	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. 5.1 Browser view on random performance tableau instance

5.3 Random Cost-Benefit performance tableaux

We provide the `RandomCBPerformanceTableau` class for generating random *Costs* versus *Benefits* organized performance tableaux following the directives below:

- We distinguish three types of decision actions: *cheap*, *neutral* and *expensive* ones with an equal proportion of 1/3. We also distinguish two types of weighted criteria: *Costs* criteria to be *minimized*, and *Benefits* criteria to be *maximized*; in the proportions 1/3 respectively 2/3.
- Random performances on each type of criteria are drawn, either from an ordinal scale [0; 10], or from a cardinal scale [0.0; 100.0], following a parametric triangular law of mode: 30% performance for cheap, 50% for neutral, and 70% performance for expensive decision actions, with constant probability repartition 0.5 on each side of the respective mode.
- Costs criteria use mostly cardinal scales (3/4), whereas Benefits criteria use mostly ordinal scales (2/3).
- The sum of weights of the Costs criteria by default equals the sum weights of the Benefits criteria: `weighDistribution = 'equiobjectives'`.

- On cardinal criteria, both of cost or of benefit type, we observe following constant preference discrimination quantiles: 5% indifferent situations, 90% strict preference situations, and 5% veto situation.

Parameters:

- If `numberOfActions == None`, a uniform random number between 10 and 31 of cheap, neutral or advantageous actions (equal 1/3 probability each type) actions is instantiated;
- If `numberOfCriteria == None`, a uniform random number between 5 and 21 of cost or benefit criteria (1/3 respectively 2/3 probability) is instantiated;
- `weightDistribution := 'equiobjectives'—'fixed'—'random'—'equisignificant'` (default = 'equisignificant');
- default `weightScale` for 'random' weight distribution is $1 - \text{numberOfCriteria}$;
- All cardinal criteria are evaluated with decimals between 0.0 and 100.0 whereas ordinal criteria are evaluated with integers between 0 and 10.
- `commonThresholds` is obsolete. Preference discrimination is specified as percentiles of concerned performance differences (see below).
- `commonPercentiles := 'ind':5, 'pref':10, 'veto':95` are expressed in percents (reversed for vetoes) and only concern cardinal criteria.
- `missingDataProbability := 0.0 ≤ float ≤ 1.0` ; probability of missing performance evaluation on a criterion for an alternative (default 0.025).
- NA := `Decimal` (default = -999); missing data symbol.

Minimal number of decision actions required is 3 !

Example Python session:

Listing 5.2 Generating a random Cost-Benefit performance tableau

```

1 >>> from randomPerfTabs import RandomCBPerformanceTableau
2 >>> t = RandomCBPerformanceTableau(
3 ...     numberOfActions=7,\ \
4 ...     numberOfCriteria=5,\ \
5 ...     weightDistribution='equiobjectives',\ \
6 ...
7 ...     commonPercentiles={'ind':0.05,'pref':0.10,'veto':0.95},\ \
8 ...     seed=100)
9
9 >>> t.showActions()
10 *----- show decision action -----*
11     key: a1
12         short name: a1
13         name:      random cheap decision action
14     key: a2
15         short name: a2
16         name:      random neutral decision action
17 ...
18     key: a7
19         short name: a7

```

```

20      name:      random advantageous decision action
21 >>> t.showCriteria()
22      *---- criteria ----*
23      g1 'random ordinal benefit criterion'
24          Scale = (0, 10)
25          Weight = 2
26          ...
27      g2 'random cardinal cost criterion'
28          Scale = (0.0, 100.0)
29          Weight = 3
30          Threshold ind : 1.76 + 0.00x ; percentile: 9.5
31          Threshold pref : 2.16 + 0.00x ; percentile: 14.3
32          Threshold veto : 73.19 + 0.00x ; percentile: 95.2
33          ...

```

In the example above, we may notice the three types of decision actions (:num-ref:‘randomCBPerformanceTableau’ Lines 10-20), as well as the two types (Lines 22-32) of criteria with either an **ordinal** or a **cardinal** performance measuring scale. 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 considered as **considerably large**. More statistics about the generated performances is available as follows.

```

1 >>> t.showStatistics()
2      ----- Performance tableau summary statistics -----
3      Instance name      : randomCBperftab
4      #Actions           : 7
5      #Criteria          : 5
6      Criterion name    : g1
7          Criterion weight   : 2
8          criterion scale   : 0.00 - 10.00
9          mean evaluation   : 5.14
10         standard deviation: 2.64
11         maximal evaluation: 8.00
12         quantile Q3 (x_75): 8.00
13         median evaluation : 6.50
14         quantile Q1 (x_25): 3.50
15         minimal evaluation: 1.00
16         mean absolute difference: 2.94
17         standard difference deviation: 3.74
18      Criterion name    : g2
19          Criterion weight   : 3
20          criterion scale   : -100.00 - 0.00
21          mean evaluation   : -49.32
22          standard deviation: 27.59
23          maximal evaluation: 0.00
24          quantile Q3 (x_75): -27.51
25          median evaluation : -35.98
26          quantile Q1 (x_25): -54.02
27          minimal evaluation: -91.87
28          mean absolute difference: 28.72
29          standard difference deviation: 39.02
30          ...

```

A (potentially ranked) colored heatmap with 5 color levels is also provided.

```
1 >>> t.showHTMLPerformanceHeatmap(colorLevels=5, rankingRule=None)
```

Heatmap of performance tableau

criteria	g3	g2	g5	g4	g1
weights	3	3	2	2	2
a1	-33.99	-17.92	3.00	26.68	1.00
a2	-77.77	-30.71	6.00	66.35	8.00
a3	-69.84	-41.65	8.00	53.43	8.00
a4	-16.99	-39.49	2.00	18.62	2.00
a5	-74.85	-91.87	7.00	83.09	6.00
a6	-24.91	-32.47	9.00	79.24	7.00
a7	-7.44	-91.11	7.00	48.22	4.00

Color legend:

quantile	0.20%	0.40%	0.60%	0.80%	1.00%
----------	-------	-------	-------	-------	-------

Fig. 5.2 Unranked heatmap of a random Cost-Benefit performance tableau

Such a performance tableau may be stored and re-accessed as follows.

```
1 >>> t.save('temp')
2     ----- saving performance tableau in XMCDa 2.0 format
3     -----
4     File: temp.py saved !
5 >>> from perfTabs import PerformanceTableau
6 >>> t = PerformanceTableau('temp')
```

If needed for instance in an R session, a CSV version of the performance tableau may be created as follows.

```
1 >>> t.saveCSV('temp')
2     * --- Storing performance tableau in CSV format in file
3     temp.csv
```

```
1 ...\\% less temp.csv
2 "actions","g1","g2","g3","g4","g5"
3 "a1",1.00,-17.92,-33.99,26.68,3.00
4 "a2",8.00,-30.71,-77.77,66.35,6.00
5 "a3",8.00,-41.65,-69.84,53.43,8.00
6 "a4",2.00,-39.49,-16.99,18.62,2.00
7 "a5",6.00,-91.87,-74.85,83.09,7.00
8 "a6",7.00,-32.47,-24.91,79.24,9.00
9 "a7",4.00,-91.11,-7.44,48.22,7.00
```

5.4 Random three objectives performance tableaux

We provide the `Random3ObjectivesPerformanceTableau` class for generating random performance tableaux concerning potential public policies evaluated with respect to three preferential decision objectives taking respectively into account *economical*, *societal* as well as *environmental* aspects.

Each public policy is qualified randomly as performing **weak** (–), **fair** (~) or **good** (+) on each of the three objectives.

Generator directives are the following:

- `numberOfActions = 20` (default),
- `numberOfCriteria = 13` (default),
- `weightDistribution = 'equiobjectives'` (default) — 'random' — 'equisignificant',
- `weightScale = (1,numberOfCriteria)`: only used when random criterion weights are requested,
- `integerWeights = True` (default): False gives normalized 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',
- `commonMode = ['triangular','variable',0.5]`: random number generators of various other types ('uniform','beta') are available,
- `valueDigits = 2` (default): evaluations are encoded as Decimals,
- `missingDataProbability = 0.05` (default): random insertion of missing values with given probability,
- `NA := ;Decimal;` (default = -999); missing data symbol,
- `seed= None`.

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. Warning: Minimal number of decision actions required is 3 !

Example Python session:

Listing 5.3 Generating a random 3 Objectives performance tableau

```

1  >>> from randomPerfTabs import
2      Random3ObjectivesPerformanceTableau
3  >>> t = Random3ObjectivesPerformanceTableau(\ 
4      ...           numberOfActions=31,\ 
5      ...           numberOfCriteria=13,\ 
6      ...           weightDistribution='equiobjectives',\ 
7      ...           seed=120)
8
9  >>> t.showObjectives()
10    ----- show objectives -----
11    Eco: Economical aspect

```

```

11      ec04 criterion of objective Eco 20
12      ec05 criterion of objective Eco 20
13      ec08 criterion of objective Eco 20
14      ec11 criterion of objective Eco 20
15      Total weight: 80.00 (4 criteria)
16      Soc: Societal aspect
17          so06 criterion of objective Soc 16
18          so07 criterion of objective Soc 16
19          so09 criterion of objective Soc 16
20          so10 criterion of objective Soc 16
21          so13 criterion of objective Soc 16
22      Total weight: 80.00 (5 criteria)
23      Env: Environmental aspect
24          en01 criterion of objective Env 20
25          en02 criterion of objective Env 20
26          en03 criterion of objective Env 20
27          en12 criterion of objective Env 20
28      Total weight: 80.00 (4 criteria)

```

In Listing 5.3 above, we notice that 5 *equisignificant* criteria (g06, g07, g09, g10, g13) evaluate for instance the performance of the public policies from a **societal** point of view (Lines 16-22). 4 *equisignificant* criteria do the same from an **economical** (Lines 10-15), respectively an **environmental** point of view (Lines 23-28). The *equiobjectives* directive results hence in a balanced total weight (80.00) for each decision objective.

```

1      >>> t.showActions()
2          key: p01
3              name:      random public policy Eco+ Soc- Env+
4              profile:   {'Eco': 'good', 'Soc': 'weak', 'Env': 'good'}
5          key: p02
6          ...
7          key: p26
8              name:      random public policy Eco+ Soc+ Env-
9              profile:   {'Eco': 'good', 'Soc': 'good', 'Env': 'weak'}
10         ...
11         key: p30
12             name:      random public policy Eco- Soc- Env-
13             profile:   {'Eco': 'weak', 'Soc': 'weak', 'Env': 'weak'}
14         ...

```

Variable triangular modes (0.3, 0.5 or 0.7 of the span of the measure scale) for each objective result in different performance status for each public policy with respect to the three objectives. Policy *p01*, for instance, will probably show *good* performances wrt the *economical* and environmental aspects, and *weak* performances wrt the *societal* aspect.

For testing purposes we provide a special :py:class:`'perfTabs.PartialPerformanceTableau'` class for extracting a **partial performance tableau** from a given tableau instance. In the example blow, we may construct the partial performance tableaux corresponding to each one of the three decision objectives.

```

1      >>> from perfTabs import PartialPerformanceTableau
2      >>> teco = PartialPerformanceTableau(t, criteriaSubset=\

```

```

3     ...
4             t . objectives [ 'Eco' ][ 'criteria' ])
5
>>> tsoc = PartialPerformanceTableau(t, criteriaSubset=\
6     ...
7             t . objectives [ 'Soc' ][ 'criteria' ])
8
>>> tenv = PartialPerformanceTableau(t, criteriaSubset=\
9     ...
             t . objectives [ 'Env' ][ 'criteria' ])

```

One may thus compute a partial bipolar-valued outranking digraph for each individual objective.

```

1   >>> from outrankingDigraphs import BipolarOutrankingDigraph
2   >>> geco = BipolarOutrankingDigraph(teco)
3   >>> gsoc = BipolarOutrankingDigraph(tsoc)
4   >>> genv = BipolarOutrankingDigraph(tenv)

```

The three partial digraphs: *geco*, *gsoc* and *genv*, hence model the preferences represented in each one of the partial performance tableaux. And, we may aggregate these three outranking digraphs with an epistemic fusion operator.

```

1   >>> from digraphs import FusionLDigraph
2   >>> gfus = FusionLDigraph([geco,gsoc,genv])
3   >>> gfus . strongComponents()
4     {frozenset({ 'p30' }), 
5      frozenset({ 'p10', 'p03', 'p19', 'p08', 'p07', 'p04', 'p21', 'p20',
6                  'p13', 'p23', 'p16', 'p12', 'p24', 'p02', 'p31',
7                  'p29',
8                  'p05', 'p09', 'p28', 'p25', 'p17', 'p14', 'p15',
9                  'p06',
10                 'p01', 'p27', 'p11', 'p18', 'p22 }), 
11     frozenset({ 'p26' })}
12 >>> from digraphs import StrongComponentsCollapsedDigraph
13 >>> scc = StrongComponentsCollapsedDigraph(gfus)
14 >>> scc . showActions()
15   ----- show digraphs actions -----
16   key: frozenset({ 'p30' })
17   short name: Scc_1
18   name: _p30_
19   comment: collapsed strong component
20   key: frozenset({ 'p10', 'p03', 'p19', 'p08', 'p07', 'p04',
21           'p21', 'p20', 'p13',
22           'p23', 'p16', 'p12', 'p24', 'p02', 'p31',
23           'p29', 'p05', 'p09', 'p28', 'p25',
24           'p17', 'p14', 'p15', 'p06', 'p01', 'p27',
25           'p11', 'p18', 'p22 })
26   short name: Scc_2
27   name:
28           _p10_p03_p19_p08_p07_p04_p21_p20_p13_p23_p16_p12_p24_p02_p31_ \
29           p29_p05_p09_p28_p25_p17_p14_p15_p06_p01_p27_p11_p18_p22_
30   comment: collapsed strong component
31   key: frozenset({ 'p26' })
32   short name: Scc_3
33   name: _p26_
34   comment: collapsed strong component

```

A graphviz drawing illustrates the apparent preferential links between the strong components.

```

1  >>> scc.exportGraphViz('scFusionObjectives')
2  ----- exporting a dot file for GraphViz tools -----
3  Exporting to scFusionObjectives.dot
4  dot -Grankdir=BT -Tpng scFusionObjectives.dot -o
   scFusionObjectives.png

```

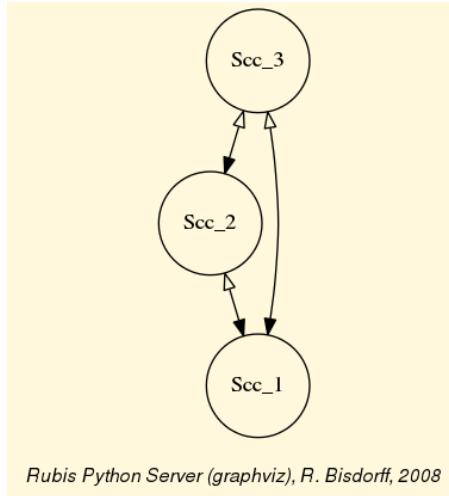


Fig. 5.3 Strong components digraph. Public policy *p26* (Eco+ Soc+ Env-) appears dominating the other policies, whereas policy *p30* (Eco- Soc- Env-) appears to be dominated by all the others.

5.5 Random academic performance tableaux

The RandomAcademicPerformanceTableau class generates temporary performance tableaux with random grades for a given number of students in different courses (see Lecture 4: *Grading*, [Algorithmic decision Theory Course](#))

Parameters:

- number of students,
- number of courses,
- weightDistribution := 'equisignificant' — 'random' (default),
- weightScale := (1, 1 — numberofCourses (default when random)),
- IntegerWeights := Boolean (True = default),
- commonScale := (0,20) (default),
- ndigits := 0,
- WithTypes := Boolean (False = default),
- commonMode := ('triangular', xm=14, r=0.25) (default),
- commonThresholds := 'ind':(0,0), 'pref':(1,0) (default),

- `missingDataProbability := 0.0` (default),
- `NA := \Decimal{.}` (default = -999); missing data symbol.

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

Listing 5.4 Generating a random academic performance tableau

```

1  >>> from randomPerfTabs import
2      RandomAcademicPerformanceTableau
3  >>> t = RandomAcademicPerformanceTableau(\ \
4      ...           numberOfStudents=11,\ \
5      ...           numberOfCourses=7, missingDataProbability=0.03,\ \
6      ...           WithTypes=True , seed=100)
7
8  >>> t
9  *----- PerformanceTableau instance description -----*
10 Instance class   : RandomAcademicPerformanceTableau
11 Seed             : 100
12 Instance name   : randstudPerf
13 # Actions        : 11
14 # Criteria       : 7
15 Attributes       : [ 'randomSeed', 'name', 'actions',
16                      'criteria', 'evaluation',
17                      'weightPreorder' ]
18 >>> t.showPerformanceTableau()
19 *----- performance tableau -----*
20 Courses | 'm1'  'm2'  'm3'  'm4'  'm5'  'm6'  'm7'
21 ECTS    | 2     1     3     4     1     1     5
22
23 |-----|
24 | 's01f' | 12    13    15    08    16    06    15
25 | 's02g' | 10    15    20    11    14    15    18
26 | 's03g' | 14    12    19    11    15    13    11
27 | 's04f' | 13    15    12    13    13    10    06
28 | 's05e' | 12    14    13    16    15    12    16
29 | 's06g' | 17    13    10    14    NA    15    13
30 | 's07e' | 12    12    12    18    NA    13    17
31 | 's08f' | 14    12    09    13    13    15    12
32 | 's09g' | 19    14    15    13    09    13    16
33 | 's10g' | 10    12    14    17    12    16    09
34 | 's11w' | 10    10    NA    10    10    NA    08
35
36 >>> t.weightPreorder
37 [[ 'm2' , 'm5' , 'm6' ], [ 'm1' ], [ 'm3' ], [ 'm4' ], [ 'm7' ]]

```

The example tableau, generated for instance above with `*missingDataProbability* = 0.03`, `*WithTypes* = True` and `*seed* = 100` (see :numref:`academicPerformanceTableau` Lines 2-5), results in a set of two excellent (`*s05*`, `*s07*`), five good (`*s02*`, `*s03*`,

s06, *s09*, *s10*), three fair (*s01*, *s04*, *s08*) and one weak (*s11*) student performances. Notice that six students get a grade below the course validating threshold 10 and we observe four missing grades (NA), two in course *m5* and one in course *m3* and course *m6* (see Lines 21-31).

We may show a statistical summary of the students' grades obtained in the highest weighted course, namely *m7*, followed by a performance heatmap browser view showing a global ranking of the students' performances from best to weakest.

Listing 5.5 Student performance summary statistics per course

```

1  >>> t.showCourseStatistics('m7')
2  ----- Summary performance statistics -----
3  Course name      : g7
4  Course weight    : 5
5  # Students       : 11
6  grading scale   : 0.00 - 20.00
7  # missing evaluations : 0
8  mean evaluation   : 12.82
9  standard deviation : 3.79
10 maximal evaluation : 18.00
11 quantile Q3 (x_75) : 16.25
12 median evaluation   : 14.00
13 quantile Q1 (x_25)   : 10.50
14 minimal evaluation   : 6.00
15 mean absolute difference : 4.30
16 standard difference deviation : 5.35
17 >>> t.showHTMLPerformanceHeatmap(colorLevels=5,\ 
18     ...                                         pageTitle='Ranking the students')

```

Ranking the students

criteria	g7	g4	g3	g1	g2	g5	g6
weights	+5.00	+4.00	+3.00	+2.00	+1.00	+1.00	+1.00
s07e	17.00	18.00	12.00	12.00	12.00	NA	13.00
s02g	18.00	11.00	20.00	10.00	15.00	14.00	15.00
s09g	16.00	13.00	15.00	19.00	14.00	9.00	13.00
s05e	16.00	16.00	13.00	12.00	14.00	15.00	12.00
s06g	13.00	14.00	10.00	17.00	13.00	NA	15.00
s03g	11.00	11.00	19.00	14.00	12.00	15.00	13.00
s10g	9.00	17.00	14.00	10.00	12.00	12.00	16.00
s01f	15.00	8.00	15.00	12.00	13.00	16.00	6.00
s08f	12.00	13.00	9.00	14.00	12.00	13.00	15.00
s04f	6.00	13.00	12.00	13.00	15.00	13.00	10.00
s11w	8.00	10.00	NA	10.00	10.00	10.00	NA

Color legend:

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

Fig. 5.4 Ranking the students with a performance heatmap view. The ranking shown here is produced with the default NetFlows ranking rule

With a mean marginal correlation of +0.361 (see Listing 5.4 Lines 17-) associated with a low standard deviation (0.248), the result represents a rather *fair weighted consensus* made between the individual courses' marginal rankings.

Listing 5.6 Consensus quality of the students's ranking

```

1  >>> from outrankingDigraphs import BipolarOutrankingDigraph
2  >>> g = BipolarOutrankingDigraph(t)
3  >>> t.showRankingConsensusQuality(g.computeNetFlowsRanking())
4      Consensus quality of ranking:
5          [ 's07', 's02', 's09', 's05', 's06', 's03', 's10',
6              's01', 's08', 's04', 's11']
7      criterion (weight): correlation
8
9      m7 (0.294): +0.727
10     m4 (0.235): +0.309
11     m2 (0.059): +0.291
12     m3 (0.176): +0.200
13     m1 (0.118): +0.109
14     m6 (0.059): +0.091
15     m5 (0.059): +0.073
16
17     Summary:
18         Weighted mean marginal correlation (a): +0.361
19         Standard deviation (b) : +0.248
20         Ranking fairness (a)-(b) : +0.113

```

5.6 Random linearly ranked performance tableaux

Finally, we provide the `RandomRankPerformanceTableau` class for generating multiple criteria ranked performance tableaux, i.e. on each criterion, all decision action's evaluations appear linearly ordered without ties.

This type of random performance tableau is matching the `RandomLinearVotingProfile` class provided by the `votingProfiles` module.

Parameters:

- number of actions,
- number of performance criteria,
- `weightDistribution` := 'equisignificant' — 'random' (default),
- `weightScale` := (1, 1 — `numberOfCriteria`) (default when random)),
- `integerWeights` := Boolean (True = default),
- `commonThresholds` (default) := `{'ind':(0,0), 'pref':(1,0), 'veto':(numberOfActions,0)}`. (default)

Concluding remarks an transition to Chapter 6.

Chapter 6

Computing a best choice recommendation

Abstract To be written.

6.1 What site to choose ?

A SME, specialized in printing and copy services, has to move into new offices, and its CEO has gathered seven **potential office sites** (see Table 6.1).

Table 6.1 The potential new office sites

ID	Name	Address	Comment
A	Ave	Avenue de la liberté	High standing city center
B	Bonnevoie		Industrial environment
C	Cessange		Residential suburb location
D	Dom	Dommeldange	Industrial suburb environment
E	Bel	Esch-Belval	New and ambitious urbanization far from the city
F	Fen	Fentange	Out in the countryside
G	Gar	Avenue de la Gare	Main city shopping street

Three **decision objectives** are guiding the CEO's choice:

1. *minimize* the yearly costs induced by the moving,
2. *maximize* the future turnover of the SME,
3. *maximize* the new working conditions.

The decision consequences to take into account for evaluating the potential new office sites with respect to each of the three objectives are modelled by the following **coherent family of criteria** Footnote [26].

The evaluation of the seven potential sites on each criterion are gathered in the following **performance tableau**.

Table 6.2 The coherent family of performance criteria

Objective	ID	Name	Comment
Yearly costs	C	Costs	Annual rent, charges, and cleaning
Future turnover	St	Standing	Image and presentation
Future turnover	V	Visibility	Circulation of potential customers
Future turnover	Pr	Proximity	Distance from town center
Working conditions	W	Space	Working space
Working conditions	Cf	Comfort	Quality of office equipment
Working conditions	P	Parking	Available parking facilities

Table 6.3 Performance evaluations of the potential office sites

Criterion	weight	A	B	C	D	E	F	G
Costs	45.0	35.0K€	17.8K€	6.7K€	14.1K€	34.8K€	18.6K€	12.0K€
Proximity	32.0	100	20	80	70	40	0	60
Visibility	26.0	60	80	70	50	60	0	100
Standing	23.0	100	10	0	30	90	70	20
Working space	10.0	75	30	0	55	100	0	50
Working comfort	6.0	0	100	10	30	60	80	50
Parking	3.0	90	30	100	90	70	0	80

Except the *Costs* criterion, all other criteria admit for grading a qualitative satisfaction scale from 0% (worst) to 100% (best). We may thus notice in Table 6.3 that site A is the most expensive, but also 100% satisfying the *Proximity* as well as the *Standing* criterion. Whereas the site C is the cheapest one; providing however no satisfaction at all on both the *Standing* and the *Working Space* criteria.

In Table 6.3 we may also see that the *Costs* criterion admits the highest significance (45.0), followed by the *Future turnover* criteria ($32.0 + 26.0 + 23.0 = 81.0$). The *Working conditions* criteria are the less significant ($10.0 + 6.0 + 3.0 = 19.0$). It follows that the CEO considers *maximizing the future turnover* the most important objective (81.0), followed by minimizing the yearly costs (45.0), and less important, *maximizing working conditions* (19.0).

Concerning yearly costs, we suppose that the CEO is indifferent up to a performance difference of 1000.00€, and he actually prefers a site if there is at least a positive difference of 2500.00€. The grades observed on the six qualitative criteria (measured in percentages of satisfaction) are very subjective and rather imprecise. The CEO is hence indifferent up to a satisfaction difference of 10%, and he claims a significant preference when the satisfaction difference is at least of 20%. Furthermore, a satisfaction difference of 80% represents for him a *considerably large*

performance difference, triggering the case given a polarisation of the preferential situations (see [BIS-2013]).

In view of Table 6.3, what is now the office site we may recommend to the CEO as **best choice**?

6.2 The DIGRAPH3 Performance tableau

A corresponding, `PerformanceTableau` object, saved in file `officeChoice.py` is provided in the `examples` directory of the DIGRAPH3 resources.

We may inspect its actual content with the computing resources provided by the `perfTabs` module.

Listing 6.1 Inspecting the `officeChoice` performance tableau

```

1 >>> from perfTabs import PerformanceTableau
2 >>> t = PerformanceTableau('examples/officeChoice')
3 >>> t
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 >>> t.showPerformanceTableau()
15 *---- performance tableau ----*
16 Criteria| 'C'        'Cf'    'P'     'Pr'    'St'    'V'     'W'
17 Weights | 45.00     6.00    3.00   32.00  23.00  26.00  10.00
18 -----
19 'Ave'   | -35000.00  0.00   90.00  100.00 100.00 60.00  75.00
20 'Bon'   | -17800.00 100.00  30.00  20.00  10.00  80.00  30.00
21 'Ces'   | -6700.00   10.00 100.00  80.00  0.00   70.00  0.00
22 'Dom'   | -14100.00  30.00  90.00  70.00  30.00  50.00  55.00
23 'Bel'   | -34800.00  60.00  70.00  40.00  90.00  60.00 100.00
24 'Fen'   | -18600.00  80.00  0.00   0.00   70.00  0.00   0.00
25 'Gar'   | -12000.00  50.00  80.00  60.00  20.00 100.00  50.00

```

We thus recover all the input data. To measure the actual preference discrimination we observe on each criterion, we may use the `showCriteria()` method.

Listing 6.2 Inspecting the performance criteria

```

1 >>> t.showCriteria(IntegerWeights=True)
2 *---- criteria -----*
3 C 'Costs'
4   Scale = (Decimal('0.00'), Decimal('50000.00'))
5   Weight = 45
6   Threshold ind : 1000.00 + 0.00x ; percentile: 9.5

```

```

7   Threshold pref : 2500.00 + 0.00x ; percentile: 14.3
8   Cf 'Comfort'
9   Scale = (Decimal('0.00'), Decimal('100.00'))
10  Weight = 6
11  Threshold ind : 10.00 + 0.00x ; percentile: 9.5
12  Threshold pref : 20.00 + 0.00x ; percentile: 28.6
13  Threshold veto : 80.00 + 0.00x ; percentile: 90.5
14  ...

```

On the *Costs* criterion, 9.5% of the performance differences are considered insignificant and 14.3% below the preference discrimination threshold (see Listing 6.2 lines 6-7). On the qualitative *Comfort* criterion, we observe again 9.5% of insignificant performance differences (line 11). Due to the imprecision in the subjective grading, we notice here 28.6% of performance differences below the preference discrimination threshold (Line 12). Furthermore, $100.0 - 90.5 = 9.5\%$ of the performance differences are judged *considerably large* (Line 13); 80% and more of satisfaction differences triggering in fact a polarisation of the preferential situation. Same information is available for all the other criteria.

A colorful comparison of all the performances is shown in Figure 6.1 by the **heatmap** statistics, illustrating the respective quantile class of each performance. As the set of potential alternatives is tiny, we choose here a classification into performance quintiles.

```

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

```

Heatmap of Performance Tableau 'officeChoice'

criteria	C	Pr	V	St	W	Cf	P
weights	+45.00	+32.00	+26.00	+23.00	+10.00	+6.00	+3.00
Ave	-35000.00	100.00	60.00	100.00	75.00	0.00	90.00
Bon	-17800.00	20.00	80.00	10.00	30.00	100.00	30.00
Ces	-6700.00	80.00	70.00	0.00	0.00	10.00	100.00
Dom	-14100.00	70.00	50.00	30.00	55.00	30.00	90.00
Bel	-34800.00	40.00	60.00	90.00	100.00	60.00	70.00
Fen	-18600.00	0.00	0.00	70.00	0.00	80.00	0.00
Gar	-12000.00	60.00	100.00	20.00	50.00	50.00	80.00

Color legend:

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

Fig. 6.1 Unranked heatmap of the office choice performance tableau.

Site *Ave* shows extreme and contradictory performances: 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 site *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 decision alternatives: sites *Dom* and *Gar*. Finally, site *Fen* is clearly the less satisfactory alternative of all.

To help now the CEO choosing the best site, we are going to compute pairwise outrankings (see [BIS-2013]) on the set of potential sites.

6.3 Computing the outranking digraph

For two sites x and y :

- x outranks y , denoted $(x \succsim y)$, is given when there is:
 1. A **majority** of criteria significance concordantly supporting that site x is *at least as satisfactory* as site y , and
 2. **No considerable** counter-performance observed on any discordant criterion.
- x does not outrank y , denoted $(x \not\succsim y)$, is given when there is:
 1. A **majority** of criteria concordantly supporting that site x is *not at least as satisfactory* as site y , and
 2. **No considerable** better performance observed on any discordant criterion.

The credibility of each pairwise outranking situation (see [BIS-2013]), 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**, and **negative** terms $r(x \succsim y) < 0.0$ indicate a **non-validated** outranking situation. The **median** value $r(x \succsim y) = 0.0$ represents an **indeterminate** situation (see [BIS-2004a]).

For computing such a bipolar-valued outranking digraph from the given performance tableau t , we use the `BipolarOutrankingDigraph` constructor from the `outrankingDigraphs` module. The corresponding `showHTMLRelationTable` method shows here the resulting bipolar-valued adjacency matrix in a system browser window (see Fig. 6.2).

```

1 >>> from outrankingDigraphs import BipolarOutrankingDigraph
2 >>> g = BipolarOutrankingDigraph(t)
3 >>> g.showHTMLRelationTable()

1 >>> g.computeCondorcetWinners()
2 ['D']
3 >>> g.computeWeakCondorcetWinners()
4 ['A', 'C', 'D']

```

We may get even more insight in the apparent outranking situations when we draw the corresponding **outranking digraph** (see Fig. 6.2).

```

1 >>> g.exportGraphViz('officeChoice')
2 *----- exporting a dot file for GraphViz tools -----*

```

Valued Adjacency Matrix

$r(x \leq y)$	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]

Fig. 6.2 In the resulting outranking relation we may notice that Alternative D is **positively outranking** all other potential office sites: D is a CONDORCET winner. Yet, alternatives A (the most expensive) and C (the cheapest) are **not outranked** by any other site; they are in fact **weak CONDORCET winners**.

```
3   Exporting to officeChoice.dot
4   dot -Grankdir=BT -Tpng officeChoice.dot -o officeChoice.png
```

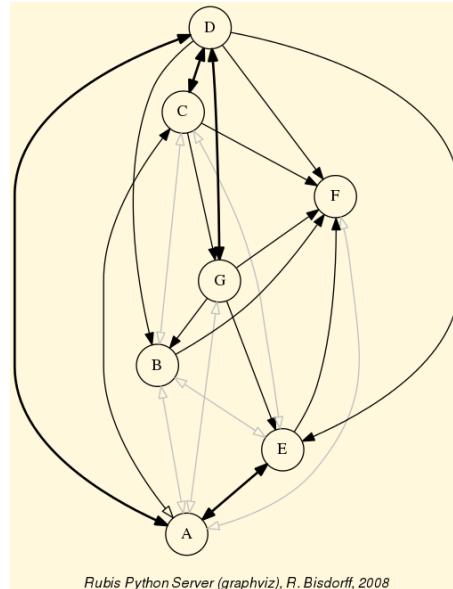


Fig. 6.3 The office choice outranking digraph.

Rubis Python Server (graphviz), R. Bisdorff, 2008

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: $r(x \not\succsim y) =$

$r(y \succsim y)$. One may furthermore check that the resulting outranking digraph g here does in fact not admit any cyclic strict preference situation.

```

1 >>> g.computeChordlessCircuits()
2     []
3 >>> g.showChordlessCircuits()
4     No circuits observed in this digraph.

```

6.4 Computing a RUBIS best choice recommendation

Following the RUBIS outranking method (see [BIS-2008]), potential best choice recommendations are determined by the outranking **prekernels** –weakly independent and strictly outranking choices– of the outranking digraph (see the tutorial on computing digraph kernels). The case given, we previously need to break open all chordless odd circuits at their weakest link.

```

1 >>> from digraphs import BrokenCocsDigraph
2 >>> bcg = BrokenCocsDigraph(g)
3 >>> bcg.brokenLinks
4     set()

```

As we observe indeed no such chordless circuits here, we may directly compute the **prekernels** of the outranking digraph g .

Listing 6.3 Computing outranking and outranked prekernels

```

1 >>> g.showPreKernels()
2     *--- Computing preKernels ---*
3     Dominant preKernels :
4         ['D']
5             independence : 1.0
6             dominance   : 0.02
7             absorbency   : -1.0
8             covering     : 1.000
9         ['B', 'E', 'C']
10            independence : 0.00
11            dominance   : 0.10
12            absorbency   : -1.0
13            covering     : 0.500
14         ['A', 'G']
15            independence : 0.00
16            dominance   : 0.78
17            absorbency   : 0.00
18            covering     : 0.700
19     Absorbent preKernels :
20         ['F', 'A']
21            independence : 0.00
22            dominance   : 0.00
23            absorbency   : 1.0
24            covering     : 0.700
25     *----- statistics -----*

```

```

26     graph name: rel_officeChoice.xml
27     number of solutions
28         dominant kernels : 3
29         absorbent kernels: 1
30     cardinality frequency distributions
31     cardinality : [0, 1, 2, 3, 4, 5, 6, 7]
32     dominant kernel : [0, 1, 1, 1, 0, 0, 0, 0]
33     absorbent kernel: [0, 0, 1, 0, 0, 0, 0, 0]
34     Execution time : 0.00018 sec.
35     Results in sets: dompreKernels and abspreKernels.

```

We notice in Listing 6.3 three potential best choice recommendations: the Condorcet winner D (Line 4), the triplet B, C and E (Line 9), and finally the pair A and G (Line 14). The best choice recommendation is now given by the **most determined** prekernel; the one supported by the most significant criteria coalition. This result is shown with the `showBestChoiceRecommendation` method. Notice that this method actually works by default on the broken chords digraph bcg .

Listing 6.4 Computing a best choice recommendation

```

1 >>> g.showBestChoiceRecommendation(CoDual=False)
2 ****
3 Rubis best choice recommendation(s) (BCR)
4 (in decreasing order of determinateness)
5 Credibility domain: [-1.00,1.00]
6 === >> potential best choice(s)
7 * choice : ['D']
8   independence : 1.00
9   dominance : 0.02
10  absorbency : -1.00
11  covering (%) : 100.00
12  determinateness (%) : 51.03
13  - most credible action(s) = { 'D': 0.02, }
14 === >> potential best choice(s)
15 * choice : ['A', 'G']
16   independence : 0.00
17   dominance : 0.78
18   absorbency : 0.00
19   covering (%) : 70.00
20   determinateness (%) : 50.00
21   - most credible action(s) = { }
22 === >> potential best choice(s)
23 * choice : ['B', 'C', 'E']
24   independence : 0.00
25   dominance : 0.10
26   absorbency : -1.00
27   covering (%) : 50.00
28   determinateness (%) : 50.00
29   - most credible action(s) = { }
30 === >> potential worst choice(s)
31 * choice : ['A', 'F']
32   independence : 0.00
33   dominance : 0.00
34   absorbency : 1.00

```

```

35     covered (%) : 70.00
36     determinateness (%) : 50.00
37     - most credible action(s) = { }
38     Execution time: 0.014 seconds

```

We notice in Listing 6.4 Line 7 above that the most significantly supported best choice recommendation is indeed the CONDORCET winner D supported by a majority of 51.03% of the criteria significance (see Line 12). Both other potential best choice recommendations, as well as the potential worst choice recommendation, are not positively validated as best, resp. worst choices. They may or may not be considered so. Alternative A, with extreme contradictory performances, appears both, in a best and a worst choice recommendation (see Lines 15 and 31) and seems hence not actually comparable to its competitors.

6.5 Computing *strict best* choice recommendations

When comparing now the performances of alternatives D and G in a pairwise perspective (see below), we notice that, with the given preference discrimination thresholds, alternative G is actually **certainly at least as good as** alternative D : $r(G \succsim D) = +145/145 = +1.0$.

Listing 6.5 Inspecting pairwise comparisons

```

1 >>> g.showPairwiseComparison('G','D')
2 *----- pairwise comparison -----*
3 Comparing actions : ('G', 'D')
4 crit. wght.   g(x)      g(y)    diff. |   ind      pref
5       concord  |
6 =====
7   'C'  45.00 -12000.00 -14100.00 +2100.00 | 1000.00 2500.00
8     +45.00  |
9   'Cf'  6.00   50.00    30.00  +20.00 |   10.00   20.00
10    +6.00  |
11   'P'   3.00   80.00    90.00  -10.00 |   10.00   20.00
12    +3.00  |
13   'Pr'  32.00   60.00    70.00  -10.00 |   10.00   20.00
14    +32.00  |
15   'St'  23.00   20.00    30.00  -10.00 |   10.00   20.00
16    +23.00  |
17   'V'   26.00  100.00    50.00  +50.00 |   10.00   20.00
18    +26.00  |
19   'W'   10.00   50.00    55.00  -5.00  |   10.00   20.00
20    +10.00  |
21 =====
22 Valuation in range: -145.00 to +145.00; global concordance:
23          +145.00

```

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

```

1 >>> g.showPairwiseComparison('C','G')
2 *----- pairwise comparison -----
3 Comparing actions : ('C','G')/('G','C')
4 crit. wght. g(x) g(y) diff. | ind. pref.
5 ('C','G')/('G','C') |
6 =====
7 'C' 45.00 -6700.00 -12000.00 +5300.00 | 1000.00 2500.00
8     +45.00/-45.00 |
9 'Cf' 6.00 10.00 50.00 -40.00 | 10.00 20.00
10    -6.00/+6.00 |
11 'P' 3.00 100.00 80.00 +20.00 | 10.00 20.00
12    +3.00/-3.00 |
13 'Pr' 32.00 80.00 60.00 +20.00 | 10.00 20.00
14    +32.00/-32.00 |
15 'St' 23.00 0.00 20.00 -20.00 | 10.00 20.00
16    -23.00/+23.00 |
17 'V' 26.00 70.00 100.00 -30.00 | 10.00 20.00
18    -26.00/+26.00 |
19 'W' 10.00 0.00 50.00 -50.00 | 10.00 20.00
20    -10.00/+10.00 |
21 =====
22 Valuation in range: -145.00 to +145.00; global concordance:
23     +15.00/-15.00
24 =====

```

To model these *strict outranking* situations, we may recompute the best choice recommendation on the **codual**, the converse (\sim) of the dual ($-$) Footnote[14], of the outranking digraph instance g (see [BIS-2013]), as follows:

Listing 6.6 Computing the strict best choice recommendation

```

1 >>> g.showBestChoiceRecommendation(\n2 ...           CoDual=True,\n3 ...           ChoiceVector=True)\n4 * --- Best and worst choice recommendation(s) ---*\n5 (in decreasing order of determinateness)\n6 Credibility domain: [-1.00,1.00]\n7 === >> potential best choice(s)\n8 * choice          : ['A', 'C', 'D']\n9   independence     : 0.00\n10  dominance        : 0.10\n11  absorbency       : 0.00\n12  covering (%)    : 41.67\n13  determinateness (%) : 50.59\n14  - characteristic vector = {\n15      'D': 0.02, 'G': 0.00, 'C': 0.00,\n16      'A': 0.00, 'F': -0.02, 'E': -0.02, 'B': -0.02, }\n17 === >> potential worst choice(s)\n18 * choice          : ['A', 'F']\n19   independence     : 0.00\n20   dominance        : -0.52\n21   absorbency       : 1.00\n22   covered (%)     : 50.00\n23   determinateness (%) : 50.00\n24   - characteristic vector = { 'G': 0.00, 'F': 0.00, 'E':\n      0.00,

```

```
25      'D': 0.00, 'C': 0.00, 'B': 0.00, 'A': 0.00, }
```

It is interesting to notice in Listing 6.6 Line 9 that the **strict best choice recommendation** consists in the set of weak Condorcet winners: *A*, *C* and *D*. In the corresponding characteristic vector (see Lines 15-16), representing the bipolar credibility degree with which each alternative may indeed be considered a best choice (see [BIS-2006a], [BIS-2006b]), we find confirmed 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. They may be potential best choice candidates besides *D*. Notice furthermore that compromise alternative *G*, while not actually included in any outranking prekernel, shows as well an indeterminate situation with respect to **being or not being** a potential best choice candidate.

We may also notice (see Line 16 and Line 19) that both alternatives *A* and *F* are reported as certainly strict outranked choices, hence as **potential worst choice recommendation**. This confirms again the global incomparability status of alternative *A* (see Fig. 6.3).

```
1 >>> gcd = ~(-g) # codual of g
2 >>> gcd.exportGraphViz(fileName='bestChoiceChoice', \
3 ...                         bestChoice=['A','C','D'], \
4 ...                         worstChoice=['F'])
5 *---- exporting a dot file for GraphViz tools -----
6 Exporting to bestOfficeChoice.dot
7 dot -Grankdir=BT -Tpng bestOfficeChoice.dot -o
   bestOfficeChoice.png
```

6.6 Weakly ordering the outranking digraph

To get a more complete insight in the overall strict outranking situations, we may use the `RankingByChoosingDigraph` constructor imported from the `transitiveDigraphs` module for computing a **ranking-by-choosing** result from the codual, i.e. the strict outranking digraph instance *gcd* (see above). If the computing node supports multiple processor cores, best and worst choosing iterations are run in parallel (see Line 3 below).

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

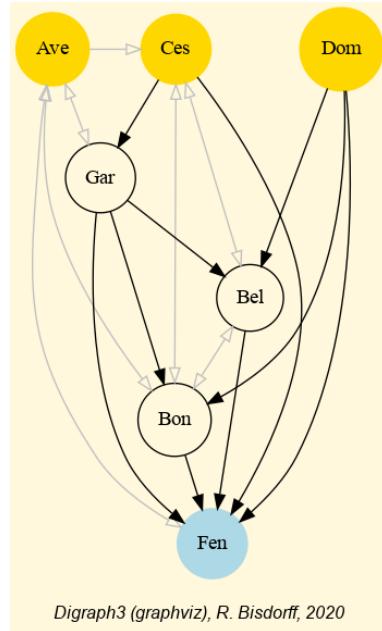


Fig. 6.4 Best office choice recommendation from strict outranking digraph.

```

13 Exporting to officeChoiceRanking.dot
14 dot -Grankdir=TB -Tpng officeChoiceRanking.dot \
15 -o officeChoiceRanking.png

```

The best choice recommendation appears hence depending on the very importance the CEO is attaching to each of the three decision objectives he is considering. In the setting here, where he considers that *maximizing the future turnover* is the most important objective followed by *minimizing the Costs* and, less important, *maximizing the working conditions*, site *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 site *A*; or if, on the contrary, *Costs* do matter a lot, moving to the cheapest alternative *C* could definitely represent a more convincing recommendation.

It might be worth, as an **exercise**, to modify these criteria significance weights in the `officeChoice.py` data file in such a way that

- all criteria under an objective appear *equi-significant*, and
- all three decision objectives are considered *equally important*.

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

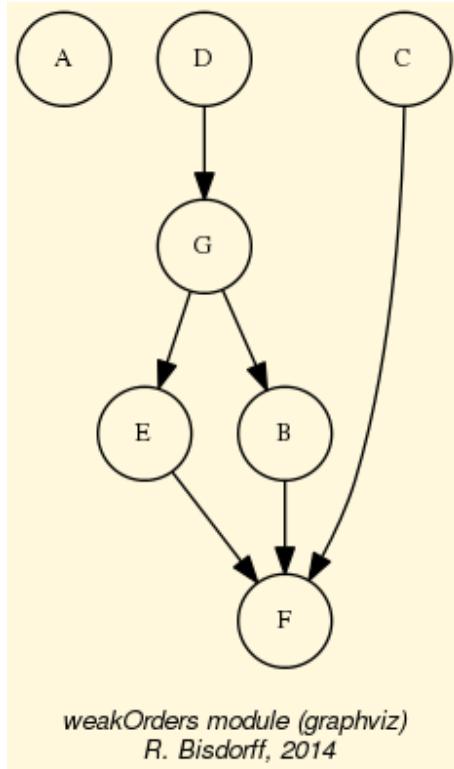


Fig. 6.5 In this **ranking-by-choosing** method, where we operate the *epistemic fusion* of iterated (strict) best and worst choices, compromise alternative *D* is now ranked before compromise alternative *G*. The overall partial ordering result shows again the important fact that the most expensive site *A*, and the cheapest site *C*, both appear incomparable with most of the other alternatives, as is apparent from the Hasse diagram of the ranking-by-choosing result here.

Chapter 7

Computing the winner of an election

Abstract To be written.

7.1 Linear voting profiles

The `votingProfiles` module provides resources for handling election results [ADT-L2], like the `LinearVotingProfile` class. We consider an election involving a finite set of candidates and finite set of weighted voters, who express their voting preferences in a complete linear ranking (without ties) of the candidates. The data is internally stored in two ordered dictionaries, one for the voters and another one for the candidates. The linear ballots are stored in a standard dictionary.

```
1 candidates = OrderedDict([( 'a1' , ... ) , ( 'a2' , ... ) , ( 'a3' , ... ) ,
2   ... ]
3 voters = OrderedDict([( ('v1' , { 'weight' :10}) , ( 'v2' , { 'weight' :3} ) ,
4   ... ]
5   ## each voter specifies a linearly ranked list of candidates
6   ## from the best to the worst (without ties)
7   linearBallot = {
8     'v1' : [ 'a2' , 'a3' , 'a1' , ... ] ,
9     'v2' : [ 'a1' , 'a2' , 'a3' , ... ] ,
10    ...
11  }
```

The module provides a `RandomLinearVotingProfile` class for generating random instances of the `LinearVotingProfile` class. In an interactive Python session we may obtain for the election of 3 candidates by 5 voters the following result.

Listing 7.1 Example of random linear voting profile

```
1 >>> from votingProfiles import RandomLinearVotingProfile
2 >>> v = RandomLinearVotingProfile(numberOfVoters=5,\n3   ...                               numberOfCandidates=3,\n4   ...                               RandomWeights=True)
```

```

5
6 >>> v.candidates
7 OrderedDict([('a1', {'name': 'a1'}), ('a2', {'name': 'a2'}),
8             ('a3', {'name': 'a3'})])
9 >>> v.voters
10 OrderedDict([('v1', {'weight': 2}), ('v2', {'weight': 3}),
11              ('v3', {'weight': 1}), ('v4', {'weight': 5}),
12              ('v5', {'weight': 4}))])
13 >>> v.linearBallot
14 {'v1': ['a1', 'a2', 'a3'],
15  'v2': ['a3', 'a2', 'a1'],
16  'v3': ['a1', 'a3', 'a2'],
17  'v4': ['a1', 'a3', 'a2'],
18  'v5': ['a2', 'a3', 'a1']}

```

Notice that in this random example, the five voters are weighted (see Listing 7.1 Line 6-7). Their linear ballots can be viewed with the `showLinearBallots` method.

```

1 >>> v.showLinearBallots()
2     voters(weight)      candidates rankings
3     v1(2):          ['a2', 'a1', 'a3']
4     v2(3):          ['a3', 'a1', 'a2']
5     v3(1):          ['a1', 'a3', 'a2']
6     v4(5):          ['a1', 'a2', 'a3']
7     v5(4):          ['a3', 'a1', 'a2']
8     nbr. of voters: 15

```

Editing of the linear voting profile may be achieved by storing the data in a file, edit it, and reload it again.

```

1 >>> v.save(fileName='tutorialLinearVotingProfile1')
2     *---- Saving linear profile in file:
3         <tutorialLinearVotingProfile1.py> ----*
4 >>> from votingProfiles import LinearVotingProfile
5 >>> v = LinearVotingProfile('tutorialLinearVotingProfile1')

```

7.2 Computing the winner

We may easily compute **uni-nominal votes**, i.e. how many times a candidate was ranked first, and see who is consequently the **simple majority** winner(s) in this election.

```

1 >>> v.computeUninominalVotes()
2     {'a2': 2, 'a1': 6, 'a3': 7}
3 >>> v.computeSimpleMajorityWinner()
4     ['a3']

```

As we observe no absolute majority (8/15) of votes for any of the three candidate, we may look for the **instant runoff** winner instead (see [ADT-L2]).

Listing 7.2 Example Instant Run Off Winner

```

1 >>> v.computeInstantRunoffWinner(Comments=True)
2   Half of the Votes = 7.50
3     ==> stage = 1
4       remaining candidates [ 'a1' , 'a2' , 'a3' ]
5       uninominal votes { 'a1': 6, 'a2': 2, 'a3': 7}
6       minimal number of votes = 2
7       maximal number of votes = 7
8       candidate to remove = a2
9       remaining candidates = [ 'a1' , 'a3' ]
10    ==> stage = 2
11    remaining candidates [ 'a1' , 'a3' ]
12    uninominal votes { 'a1': 8, 'a3': 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: [ 'a1' ]

```

In stage 1, no candidate obtains an absolute majority of votes. Candidate a_2 obtains the minimal number of votes ($2/15$) and is, hence, eliminated. In stage 2, candidate a_1 obtains an absolute majority of the votes ($8/15$) and is eventually elected (see Listing 7.2).

We 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)**.

Listing 7.3 Example of *Borda* rank scores

```

1 >>> v.computeRankAnalysis()
2 { 'a2': [2, 5, 8], 'a1': [6, 9, 0], 'a3': [7, 1, 7] }
3 >>> v.computeBordaScores()
4 OrderedDict([
5   ('a1', { 'BordaScore': 24, 'averageBordaScore': 1.6}),
6   ('a3', { 'BordaScore': 30, 'averageBordaScore': 2.0}),
7   ('a2', { 'BordaScore': 36, 'averageBordaScore': 2.4})])
8 >>> v.computeBordaWinners()
9 [ 'a1' ]

```

Candidate a_1 obtains the minimal *Borda* score, followed by candidate a_3 and finally candidate a_2 (see Listing 7.3). The corresponding **Borda rank analysis table** may be printed out with a corresponding `show` command.

Listing 7.4 Rank analysis example

```

1 >>> v.showRankAnalysisTable()
2 *----- Borda rank analysis tableau -----*
3 candi- | alternative-to-rank |      Borda
4 dates  |      1      2      3 | score  average
5 -----
6   'a1'  |      6      9      0 | 24/15  1.60
7   'a3'  |      7      1      7 | 30/15  2.00
8   'a2'  |      2      5      8 | 36/15  2.40

```

In our randomly generated election results, we are lucky: The instant runoff winner and the *Borda* winner both is candidate a_1 (see Listings 7.2 and 7.3). However,

we could also follow the *Marquis de Condorcet*'s advice, and compute the **majority margins** obtained by voting for each individual pair of candidates.

7.3 The Condorcet winner

For instance, candidate a_1 is ranked four times before and once behind candidate a_2 . Hence the corresponding **majority margin** $M(a_1, a_2)$ is $4 - 1 = +3$. These *majority margins* define on the set of candidates what we call the **majority margins digraph**. The `MajorityMarginsDigraph` class (a specialization of the `Digraph` class) is available for handling such kind of digraphs.

Listing 7.5 Example of *Majority Margins* digraph

```

1 >>> from votingProfiles import MajorityMarginsDigraph
2 >>> mmdg = MajorityMarginsDigraph(v, IntegerValuation=True)
3 >>> mmdg
4 *----- Digraph instance description -----
5 Instance class      : MajorityMarginsDigraph
6 Instance name       : rel_randomLinearVotingProfile1
7 Digraph Order       : 3
8 Digraph Size        : 3
9 Valuation domain    : [-15.00;15.00]
10 Determinateness (%) : 64.44
11 Attributes          : [ 'name', 'actions', 'voters',
12                           'ballot', 'valuationdomain',
13                           'relation', 'order',
14                           'gamma', 'notGamma' ]
15 >>> mmdg.showAll()
16 *----- show detail -----
17 Digraph           : rel_randomLinearVotingProfile1
18 *---- Actions ----*
19   [ 'a1', 'a2', 'a3' ]
20 *---- Characteristic valuation domain ----*
21   { 'max': Decimal('15.0'), 'med': Decimal('0'),
22     'min': Decimal('-15.0'), 'hasIntegerValuation': True }
23 *---- majority margins ----*
24   M(x,y) | 'a1'  'a2'  'a3'
25   -----|-----
26   'a1'  | 0     11    1
27   'a2'  | -11   0     -1
28   'a3'  | -1     1     0
29 Valuation domain: [-15;+15]
```

Notice that in the case of linear voting profiles, majority margins always verify a zero sum property: $M(x,y) + M(y,x) = 0$ for all candidates x and y (see Listing 7.5 Lines 26-28). This is not true in general for arbitrary voting profiles. The *majority margins* digraph of linear voting profiles defines in fact a *weak tournament* and belongs, hence, to the class of *self-codual* bipolar-valued digraphs (Footnote[13]).

Now, a candidate x , showing a positive majority margin $M(x,y)$, is beating candidate y with an absolute majority in a pairwise voting. Hence, a candidate showing

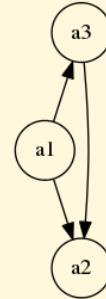
only positive terms in her row in the *majority margins* digraph relation table, beats all other candidates with absolute majority of votes. Condorcet recommends to declare this candidate (is always unique, why?) the winner of the election. Here we are lucky, it is again candidate a_1 who is hence the **Condorcet winner** (see Listing 7.5 Line 26).

```
1 >>> mmdg.computeCondorcetWinners()
2 [ 'a1' ]
```

By seeing the majority margins like a *bipolar-valued characteristic function* of a global preference relation defined on the set of candidates, we may use all operational resources of the generic `Digraph` class (see Chapter 2), and especially its `exportGraphViz` method Footnote [1], for visualizing an election result.

```
1 >>> mmdg.exportGraphViz(fileName='tutorialLinearBallots')
2 *----- exporting a dot file for GraphViz tools -----
3 Exporting to tutorialLinearBallots.dot
4 dot -Grankdir=BT -Tpng tutorialLinearBallots.dot \
5 -o tutorialLinearBallots.png
```

Fig. 7.1 Visualizing an election result. In the Figure we notice that the *majority margins* digraph from our example linear voting profile gives a linear order of the candidates: $a_1 > a_3 > a_2$, the same actually as given by the *Borda* scores (see Listing 7.4). This is by far not given in general. Usually, when aggregating linear ballots, there appear cyclic social preferences.



Rubis Python Server (graphviz), R. Bisdorff, 2008

7.4 Cyclic social preferences

Let us consider for instance the following linear voting profile and construct the corresponding majority margins digraph.

Listing 7.6 Example of cyclic social preferences

```
1 >>> v.showLinearBallots()
2     voters(weight)      candidates rankings
3     v1(1):      [ 'a1', 'a3', 'a5', 'a2', 'a4' ]
4     v2(1):      [ 'a1', 'a2', 'a4', 'a3', 'a5' ]
5     v3(1):      [ 'a5', 'a2', 'a4', 'a3', 'a1' ]
6     v4(1):      [ 'a3', 'a4', 'a1', 'a5', 'a2' ]
```

```

7     v5(1) :      [ 'a4' , 'a2' , 'a3' , 'a5' , 'a1' ]
8     v6(1) :      [ 'a2' , 'a4' , 'a5' , 'a1' , 'a3' ]
9     v7(1) :      [ 'a5' , 'a4' , 'a3' , 'a1' , 'a2' ]
10    v8(1) :      [ 'a2' , 'a4' , 'a5' , 'a1' , 'a3' ]
11    v9(1) :      [ 'a5' , 'a3' , 'a4' , 'a1' , 'a2' ]
12 >>> mmdg = MajorityMarginsDigraph(v)
13 >>> mmdg.showRelationTable()
14 * ----- Relation Table -----
15   S | 'a1' 'a2' 'a3' 'a4' 'a5'
16   -----
17   'a1' | - 0.11 -0.11 -0.56 -0.33
18   'a2' | -0.11 - 0.11 0.11 -0.11
19   'a3' | 0.11 -0.11 -  -0.33 -0.11
20   'a4' | 0.56 -0.11 0.33 - 0.11
21   'a5' | 0.33 0.11 0.11 -0.11 -

```

Now, we cannot find any completely positive row in the relation table (see Listing 7.6 Lines 17 -). 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 at a graphviz drawing of this *majority margins* digraph, we may observe **cyclic** preferences, like ($a_1 > a_2 > a_3 > a_1$) for instance.

```

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

```

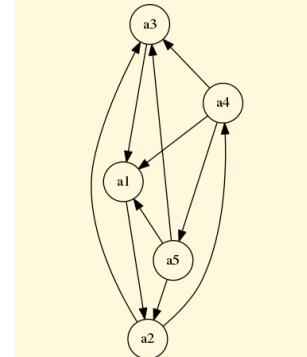


Fig. 7.2 Cyclic social preferences.

Rubis Python Server (graphviz), R. Bisдорff, 2008

But, there may be many cycles appearing in a *majority margins* digraph, and, we may detect and enumerate all minimal chordless circuits in a Digraph instance with the `computeChordlessCircuits` method.

```

1 >>> mmdg.computeChordlessCircuits()
2 [([ 'a2' , 'a3' , 'a1' ] , frozenset({ 'a2' , 'a3' , 'a1' })), 
3 ([ 'a2' , 'a4' , 'a5' ] , frozenset({ 'a2' , 'a5' , 'a4' })), 
4 ([ 'a2' , 'a4' , 'a1' ] , frozenset({ 'a2' , 'a1' , 'a4' }))]

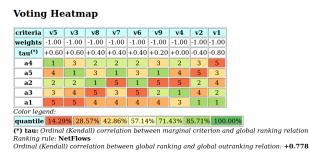
```

Condorcet's approach for determining the winner of an election is hence *not decisive* in all circumstances and we need to exploit more sophisticated approaches for finding the winner of the election on the basis of the majority margins of the given linear ballots (see Chapters 7, 8 and [BIS-2008]).

Many more tools for exploiting voting results are available like the browser heat map view on voting profiles. The number of voters is usually much larger than the number of candidates. In that case, it is better to generate a transposed *voters* \times *candidates* view (see Line 2 below)

```
1 >>> v.showHTMLVotingHeatmap(rankingRule='NetFlows',
2 ... Transposed=False)
```

Fig. 7.3 Visualizing a linear voting profile in a heatmap format. 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*, i.e. goes from lowest (best) rank to highest (worst) rank.



It worthwhile noticing that the compromise *NetFlows* ranking $a_4 > a_5 > a_2 > a_1 > a_3$, shown in this heatmap (see Fig. 7.3) results in an optimal *ordinal correlation* index of +0.778 with the pairwise majority voting margins. == $\zeta\zeta$ (see Chapters on Ordinal Correlation equals Relational Equivalence and Ranking-Tutorial-label).

7.5 On generating realistic random linear voting profiles

By default, the `RandomLinearVotingProfile` class generates random linear voting profiles where every candidates has the same uniform probabilities 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.

In reality, political election data appear quite different. There usually will be different favorite and marginal candidates for each political party. To simulate these aspects into our random generator, we are using two random exponentially distributed polls of the candidates and consider a bipartisan political landscape with a certain random balance (default theoretical party repartition = 0.50) between the two sets of potential party supporters. A certain theoretical proportion (default = 0.1) will not support any party.

Let us generate such a linear voting profile for an election with 1000 voters and 15 candidates.

Listing 7.7 Generating a linear voting profile with random polls

```

1 >>> from votingProfiles import RandomLinearVotingProfile
2 >>> lvp = RandomLinearVotingProfile(\ 
3 ...     numberOfCandidates=15,
4 ...     numberOfVoters=1000,
5 ...     WithPolls=True,
6 ...     partyRepartition=0.5,
7 ...     other=0.1,
8 ...     seed=0.9189670954954139)
9
10 >>> lvp
*----- VotingProfile instance description -----
11 Instance class : RandomLinearVotingProfile
12 Instance name  : randLinearProfile
13 Candidates    : 15
14 Voters        : 1000
15 Attributes   : [ 'name', 'seed', 'candidates',
16 ...                 'voters', 'RandomWeights',
17 ...                 'sumWeights', 'poll1', 'poll2',
18 ...                 'bipartisan', 'linearBallot', 'ballot']
19
20 >>> lvp.showRandomPolls()
21 Random repartition of voters
22 Party_1 supporters : 460 (46.0%)
23 Party_2 supporters : 436 (43.6%)
24 Other voters       : 104 (10.4%)
*----- random polls -----
25
26 Party -1(46.0%) | Party -2(43.6%) | expected
27
28   a06 : 19.91% | a11 : 22.94% | a06 : 15.00%
29   a07 : 14.27% | a08 : 15.65% | a11 : 13.08%
30   a03 : 10.02% | a04 : 15.07% | a08 : 09.01%
31   a13 : 08.39% | a06 : 13.40% | a07 : 08.79%
32   a15 : 08.39% | a03 : 06.49% | a03 : 07.44%
33   a11 : 06.70% | a09 : 05.63% | a04 : 07.11%
34   a01 : 06.17% | a07 : 05.10% | a01 : 05.06%
35   a12 : 04.81% | a01 : 05.09% | a13 : 05.04%
36   a08 : 04.75% | a12 : 03.43% | a15 : 04.23%
37   a10 : 04.66% | a13 : 02.71% | a12 : 03.71%
38   a14 : 04.42% | a14 : 02.70% | a14 : 03.21%
39   a05 : 04.01% | a15 : 00.86% | a09 : 03.10%
40   a09 : 01.40% | a10 : 00.44% | a10 : 02.34%
41   a04 : 01.18% | a05 : 00.29% | a05 : 01.97%
42   a02 : 00.90% | a02 : 00.21% | a02 : 00.51%

```

In this example (see Listing 7.7 Lines 18-), we obtain 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%, appear to be $a06$ (19.91%), $a07$ (14.27%) and $a03$ (10.02%). Whereas for Party-2 supporters, favorite candidates appear to be $a11$ (22.94%), followed by $a08$ (15.65%), $a04$ (15.07%) and $a06$ (13.4%). Being *first* choice for Party-1 supporters and *fourth* choice for Party-2 supporters, this candidate $a06$ is a natural candidate for clearly winning this election game (see Listing 7.8).

Listing 7.8 The uninominal and *Borda* election winner

```

1 >>> lvp.computeSimpleMajorityWinner()
2 [ 'a06' ]
3 >>> lvp.computeInstantRunoffWinner()
4 [ 'a06' ]
5 >>> lvp.computeBordaWinners()
6 [ 'a06' ]

```

Is it 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 105 *oriented arcs* -the *positive* majority margins- (see Listing 7.9 Lines 7-8).

Listing 7.9 A majority margins digraph constructed from a linear voting profile

```

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

```

We may visualize the resulting pairwise majority margins by showing the HTML formated version of the *mmdg* relation table in a browser view.

```

1 >>> cdg.showHTMLRelationTable(tableTitle='Pairwise majority
2   margins',
3   relationName='M(x>y)')

```

	a01	a02	a03	a04	a05	a06	a07	a08	a09	a10	a11	a12	a13	a14	a15							
a01	-	768,-138,101,479,-436,-198,-149,238,540,-268,145,-101,-101,-101,-101																				
a02		-	796,-484,100,508,-456,-172,-172,-172,-172,-172,-172,-172,-172,-172,-172,-172																			
a03			-	130,796,-160,599,-370,-80,-80,-80,-80,-80,-80,-80,-80,-80,-80,-80																		
a04				-	100,494,-160,-184,-370,-180,-280,160,136,-420,19,-42,-56,-30																	
a05					-	478,368,-590,-184,-730,-640,-472,-234,-116,-550,-442,-522,-376,-386																
a06						-	430,858,-286,370,-248,234,574,692,-554,482,566,-550,-380,-380,-380,-380															
a07							-	140,772,-8,280,-472,-234,0,-436,596,176,276,134,298,244														
a08								-	238,546,-372,-160,234,-574,-358,-436,-116,594,-126,-193,-90,-14													
a09									-	440,406,-522,-136,116,-492,-602,-396,-116,-510,-310,-442,-304,-266												
a10										-	111,333,-111,-111,-111,-111,-111,-111,-111,-111,-111,-111,-111,-111,-111,-111											
a11											-	148,722,-339,-10,-442,-156,-276,126,210,-388,-388,-265,-265,-265,-265,-265										
a12												-	50,708,-210,62,522,-482,-266,-134,194,442,-268,92,-158,166									
a13													-	202,696,-360,-56,376,-566,-384,-298,90,304,474,-109,-185,-68,-68								
a14														-	218,658,-338,-30,386,-520,-420,-244,14,266,-292,-149,-186,-68							
a15															-	Valuation domain: [-1000;1000]						

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

A complete *light green* row reveals a *Condorcet winner*, whereas a complete *light green* column reveals a *Condorcet loser*. We recover again candidate *a06* as *Condorcet* winner (Footnote[15]), whereas the obvious *Condorcet* loser is here candidate *a02*, the candidate with the lowest support in both parties (see Listing 7.7 Line 40).

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

Listing 7.10 Ranking by iterating choosing the *first* and *last* remaining candidates

```

1 >>> cdg.showRankingByChoosing()
2 Error: You must first run
3 self.computeRankingByChoosing(CoDual=False(default)|True) !
4 >>> cdg.computeRankingByChoosing()
5 >>> cdg.showRankingByChoosing()
6 Ranking by Choosing and Rejecting
7   1st first ranked ['a06']
8     2nd first ranked ['a11']
9       3rd first ranked ['a07', 'a08']
10      4th first ranked ['a03']
11        5th first ranked ['a01']
12          6th first ranked ['a13']
13            7th first ranked ['a04']
14              7th last ranked ['a12']
15                6th last ranked ['a14']
16                  5th last ranked ['a15']
17                    4th last ranked ['a09']
18                      3rd last ranked ['a10']
19                        2nd last ranked ['a05']
20                          1st last ranked ['a02']

```

Before showing the *ranking-by-choosing* result, we have to compute the iterated bipolar selection procedure (see Listing 7.10 Line 2). The first selection concerns a_06 (first) and a_02 (last), followed by a_{11} (first) opposed to a_{05} (last), and so on, until there remains at iteration step 7 a last pair of candidates, namely $[a_04, a_{12}]$ (see Lines 13-14).

Notice furthermore the first ranked candidates at iteration step 3 (see Line 9), namely the pair (a_07, a_08) . Both candidates represent indeed conjointly the *first ranked* choice. We obtain here hence a *weak ranking*, i.e. a ranking with a tie.

Let us mention that the *instant-run-off* procedure, we used before (see Listing 7.8 Line 3), when operated with a `Comments=True` parameter setting, will deliver a more or less similar *reversed linear ordering-by-rejecting* result, namely $[a_02, a_{10}, a_{14}, a_{05}, a_9, a_{13}, a_{12}, a_{15}, a_4, a_1, a_8, a_3, a_7, a_{11}, a_6]$, ordered from the *last* to the *first* choice.

Remarkable about both these *ranking-by-choosing* or *ordering-by-rejecting* results is the fact that the random voting behaviour, simulated here with the help of two discrete random variables (Footnote [16]), defined respectively by the two party polls, is rendering a ranking that is more or less in accordance with the simulated balance of the polls: -Party-1 supporters : 460; Party-2 supporters: 436 (see Listing 7.7 Lines 26-40 third column). Despite a random voting behaviour per voter, the given polls apparently show a *very strong incidence* on the eventual election result. In order to avoid any manipulation of the election outcome, public media are therefore in some countries not allowed to publish polls during the last weeks before a general election.

Mind that the specific *ranking-by-choosing* procedure, we use here on the *majority margins* digraph, operates the selection procedure by extracting at each step *initial* and *terminal* kernels, i.e. NP-hard operational problems (see tutorial on kernels and [BIS-1999]); A technique that does not allow in general to tackle voting profiles with much more than 30 candidates. The tutorial on ranking methods provides more adequate and efficient techniques for ranking from pairwise majority margins when a larger number of potential candidates is given.

Chapter 8

Ranking with multiple incommensurable criteria

Abstract To be written.

8.1 The ranking problem

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 may know their pairwise bipolar-valued *strict outranking* characteristics, i.e. $r(x \succ y)$ for all $x, y \in X$ (see CoDual-Digraph-label and [BIS-2013]).

Let us consider a didactic outranking digraph odg generated from a random Cost-Benefit performance tableau (see Section 5.3) concerning 9 decision alternatives evaluated on 13 performance criteria. We may compute the corresponding *strict outranking digraph* with a codual transform (see Section 2.6) as follows.

Listing 8.1 Random bipolar-valued strict outranking relation characteristics

```
1 >>> from randomPerfTabs import RandomCBPerformanceTableau
2 >>> t = RandomCBPerformanceTableau(numberOfActions=9, \
3 ...           numberOfCriteria=13, seed=200)
4 >>> from outrankingDigraphs import BipolarOutrankingDigraph
5 >>> g = BipolarOutrankingDigraph(t, Normalized=True)
6 >>> gcd = ~(-g) # codual digraph
7 >>> gcd.showRelationTable(ReflexiveTerms=False)
8 * ----- Relation Table -----
9 r(>) | 'a1' 'a2' 'a3' 'a4' 'a5' 'a6' 'a7' 'a8' 'a9'
10 -----|-----
11 'a1' |   -  0.00 +0.10 -1.00 -0.13 -0.57 -0.23 +0.10 +0.00
12 'a2' | -1.00   -  0.00 +0.00 -0.37 -0.42 -0.28 -0.32 -0.12
13 'a3' | -0.10  0.00   - -0.17 -0.35 -0.30 -0.17 -0.17 +0.00
14 'a4' |  0.00  0.00 -0.42   - -0.40 -0.20 -0.60 -0.27 -0.30
15 'a5' | +0.13 +0.22 +0.10 +0.40   - +0.03 +0.40 -0.03 -0.07
16 'a6' | -0.07 -0.22 +0.20 +0.20 -0.37   - +0.10 -0.03 -0.07
17 'a7' | -0.20 +0.28 -0.03 -0.07 -0.40 -0.10   - +0.27 +1.00
18 'a8' | -0.10 -0.02 -0.23 -0.13 -0.37 +0.03 -0.27   - +0.03
```

```
19   'a9' |  0.00 +0.12 -1.00 -0.13 -0.03 -0.03 -1.00 -0.03 -
```

Some ranking rules will work on the associated **Condorcet Digraph**, i.e. the corresponding *strict median cut* polarised digraph.

Listing 8.2 Median cut polarised strict outranking relation characteristics

```
1 >>> ccd = PolarisedOutrankingDigraph(gcd,\n2 ...                      level=g.valuationdomain['med'],\n3 ...                      KeepValues=False, StrictCut=True)\n4\n5 >>> ccd.showRelationTable(ReflexiveTerms=False,\n6 ...                           IntegerValues=True)\n7 *---- Relation Table ----\n8 r(>)_med | 'a1' 'a2' 'a3' 'a4' 'a5' 'a6' 'a7' 'a8' 'a9'\n9 -----|-----\n10 'a1' | - 0 +1 -1 -1 -1 -1 +1 0\n11 'a2' | -1 - +0 0 -1 -1 -1 -1 -1\n12 'a3' | -1 0 - -1 -1 -1 -1 -1 0\n13 'a4' | 0 0 -1 - -1 -1 -1 -1 -1\n14 'a5' | +1 +1 +1 +1 - +1 +1 -1 -1\n15 'a6' | -1 -1 +1 +1 -1 - +1 -1 -1\n16 'a7' | -1 +1 -1 -1 -1 -1 - +1 +1\n17 'a8' | -1 -1 -1 -1 -1 +1 -1 - - +1\n18 'a9' | 0 +1 -1 -1 -1 -1 -1 -1 -
```

Unfortunately, such crisp median-cut *Condorcet* digraphs, associated with a given strict outranking digraph, present only exceptionally a linear ordering. Usually, pairwise majority comparisons do not even render a *complete* or, at least, a *transitive* partial order. There may even frequently appear *cyclic* outranking situations (see Section 6.5).

To estimate how *difficult* this ranking problem here may be, we may have a look at the corresponding strict outranking digraph *graphviz* drawing (Footnote [1]).

```
1 >>> gcd.exportGraphViz('rankingTutorial')\n2 *---- exporting a dot file for GraphViz tools -----*\n3 Exporting to rankingTutorial.dot\n4 dot -Grankdir=BT -Tpng rankingTutorial.dot -o\n      rankingTutorial.png
```

We may compute the transitivity degree of the outranking digraph, i.e. the ratio of the difference between the number of outranking arcs and the number of transitive arcs over the difference of the number of arcs of the transitive closure minus the transitive arcs of the digraph *gcd*.

```
1 >>> gcd.computeTransitivityDegree(Comments=True)\n2 Transitivity degree of graph <codual_rel_randomCBperftab>\n3 triples x>y>z: 78, closed: 38, open: 40\n4 closed/triples = 0.487
```

With only 35% of the required transitive arcs, the strict outranking relation here is hence very far from being transitive; a serious problem when a linear ordering of the decision alternatives is looked for. Let us furthermore see if there are any cyclic outrankings.

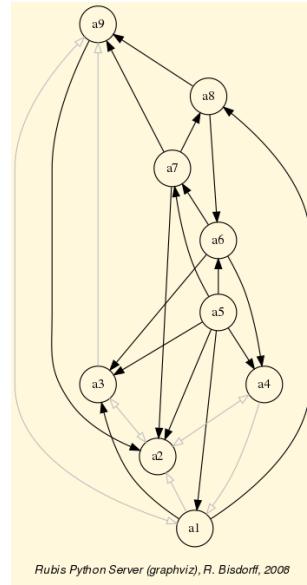


Fig. 8.1 The strict outranking relation \succsim shown here is apparently *not transitive*: for instance, alternative a_8 outranks alternative a_6 and alternative a_6 outranks a_4 , however a_8 does not outrank a_4 .

```

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

```

There is one chordless circuit detected in the given strict outranking digraph gcd , namely a_6 outranks a_7 , the latter outranks a_8 , and a_8 outranks again a_6 (see Fig. 8.1). Any potential linear ordering of these three alternatives will, in fact, always contradict somehow the given outranking relation.

Now, several heuristic ranking rules have been proposed for constructing a linear ordering which is closest in some specific sense to a given outranking relation.

The DIGRAPH3 resources provide some of the most common of these ranking rules, like *Copeland's*, *Kemeny's*, *Slater's*, *Kohler's*, *Arrow-Raynaud's* or *Tideman's* ranking rule.

8.2 The COPELAND ranking

Copeland's rule, the most intuitive one as it works well for any strict outranking relation which models in fact a linear order, works on the *median cut* strict outranking digraph ccd . The rule computes for each alternative a score resulting from the sum of the differences between the crisp **strict outranking** characteristics $r(x \succsim y)_{>0}$ and the crisp **strict outranked** characteristics $r(y \succsim x)_{>0}$ for all pairs of alternatives where y is different from x . The alternatives are ranked in decreasing order of

these COPELAND scores; ties, the case given, being resolved by a lexicographical rule.

Listing 8.3 Computing a COPELAND Ranking

```

1 >>> from linearOrders import CopelandRanking
2 >>> cop = CopelandRanking(gcd, Comments=True)
3 Copeland decreasing scores
4     a5 : 12
5     a1 : 2
6     a6 : 2
7     a7 : 2
8     a8 : 0
9     a4 : -3
10    a9 : -3
11    a3 : -5
12    a2 : -7
13 Copeland Ranking:
14 ['a5', 'a1', 'a6', 'a7', 'a8', 'a4', 'a9', 'a3', 'a2']

```

Alternative a_5 obtains here the best COPELAND score (+12), followed by alternatives a_1 , a_6 and a_7 with same score (+2); following the lexicographic rule, a_1 is hence ranked before a_6 and a_6 before a_7 . Same situation is observed for a_4 and a_9 with a score of -3 (see Listing 8.3 Lines 4-12).

COPELAND's ranking rule appears in fact **invariant** under the codual transform (see Section 2.6) and renders a same linear order indifferently from digraphs g or gcd . The resulting ranking (see Listing 8.3 Line 14) is rather correlated (+0.463) with the given pairwise outranking relation in the ordinal KENDALL sense (see Chapter on Correlation).

Listing 8.4 Checking the quality of the COPELAND Ranking

```

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

```

With an epistemic determination level of 0.230, the *extended* KENDALL τ index (see [BIS-2012]) is in fact computed on $61.5\%(100.0 \times (1.0 + 0.23)/2)$ of the pairwise strict outranking comparisons. Furthermore, the bipolar-valued *relational equivalence* characteristics between the strict outranking relation and the COPELAND ranking equals +0.107, i.e. a *majority* of 55.35% of the criteria significance supports the relational equivalence between the given strict outranking relation and the corresponding COPELAND ranking.

The COPELAND scores deliver actually only a unique *weak ranking*, i.e. a ranking with potential ties. This weak ranking may be constructed with the `WeakCopelandOrder` class.

Listing 8.5 Checking the quality of the COPELAND Ranking

```

1 >>> from transitiveDigraphs import WeakCopelandOrder
2 >>> wcop = WeakCopelandOrder(g)

```

```

3 >>> wcop.showRankingByChoosing()
4 Ranking by Choosing and Rejecting
5   1st ranked ['a5']
6   2nd ranked ['a1', 'a6', 'a7']
7   3rd ranked ['a8']
8   3rd last ranked ['a4', 'a9']
9   2nd last ranked ['a3']
10  1st last ranked ['a2']

```

We recover in Listing 8.5 above, the ranking with ties delivered by the COPELAND scores (see Listing 8.3). We may draw its corresponding Hasse diagram.

```

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

```

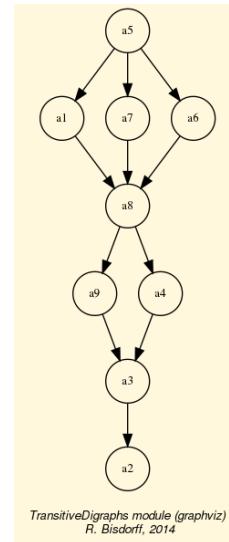


Fig. 8.2 A weak Copeland ranking.

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

8.3 The NETFLOWS ranking

The valued version of the COPELAND rule, called NETFLOWS rule, computes for each alternative x a **net flow** score, i.e. the sum of the differences between the *strict outranking* characteristics $r(x \succ y)$ and the *strict outranked* characteristics $r(y \succ x)$ for all pairs of alternatives where y is different from x .

Listing 8.6 Computing a NETFLOWS ranking

```

1 >>> from linearOrders import NetFlowsRanking
2 >>> nf = NetFlowsRanking(gcd, Comments=True)
3   Net Flows :
4     a5 : 3.600
5     a7 : 2.800
6     a6 : 1.300
7     a3 : 0.033
8     a1 : -0.400
9     a8 : -0.567
10    a4 : -1.283
11    a9 : -2.600
12    a2 : -2.883
13   NetFlows Ranking:
14   ['a5', 'a7', 'a6', 'a3', 'a1', 'a8', 'a4', 'a9', 'a2']
15 >>> cop.copelandRanking
16   ['a5', 'a1', 'a6', 'a7', 'a8', 'a4', 'a9', 'a3', 'a2']

```

It is worthwhile noticing again, that similar to the COPELAND ranking rule seen before, the NETFLOWS ranking rule is also **invariant** under the codual transform (see Secion 2.6) and delivers again the same ranking result indifferently from digraphs g or gcd (see Listing 8.6 Line 14).

In our example here, the NETFLOWS scores actually deliver a ranking *without ties* which is rather different from the one delivered by COPELAND's rule (see Line 16). It may happen, however, that we obtain, as with the COPELAND scores above, only a ranking with ties, which may then be resolved again by following a lexicographic rule. In such cases, it is possible to construct again a *weak ranking* with the corresponding `WeakNetFlowsOrder` class.

The NETFLOWS ranking result appears to be slightly better correlated (+0.638) with the given outranking relation than its crisp cousin, the COPELAND ranking (see Lines 4-6 below).

Listing 8.7 Checking the quality of the NETFLOWS Ranking

```

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

```

Indeed, the extended KENDALL τ index of +0.638 leads to a bipolar-valued *relational equivalence* characteristics of +0.147, i.e. a majority of 57.35% of the criterial significance supports the relational equivalence between the given outranking digraphs g or gcd and the corresponding NETFLOWS ranking. This lesser ranking performance of the COPELAND rule stems in this example essentially from the *weakness* of the actual ranking result and our subsequent *arbitrary* lexicographic resolution of the many ties given by the COPELAND scores (see Fig. 8.2).

To appreciate now the more or less correlation of both the COPELAND and the NETFLOWS rankings with the underlying pairwise outranking relation, it is useful to consider KEMENY's and SLATER's **best fitting** ranking rules.

8.4 KEMENY rankings

A KEMENY ranking is a linear ranking without ties which is *closest*, in the sense of the ordinal KENDALL distance (see [BIS-2012]), to the given valued outranking digraphs g or gcd . This rule is also *invariant* under the codual transform.

Listing 8.8 Computing a KEMENY ranking

```

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

```

So, $+0.779$ represents the *highest possible* ordinal correlation (fitness) any potential linear ranking can achieve with the given pairwise outranking digraph (see Listing 8.8 Lines 7-10).

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 12.9.

Listing 8.9 Optimal KEMENY rankings

```

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

```

We may visualize the partial order defined by the epistemic disjunction (see Section 2.5) of both optimal KEMENY rankings by using the `RankingsFusion` class as follows:

Listing 8.10 Computing the epistemic disjunction of all optimal KEMENY rankings

```

1 >>> from transitiveDigraphs import RankingsFusion
2 >>> wke = RankingsFusion(ke, ke.maximalRankings)
3 >>> wke.exportGraphViz(fileName='tutorialKemeny')
4   ----- exporting a dot file for GraphViz tools -----
5   Exporting to tutorialKemeny.dot
6   dot -Grankdir=TB -Tpng tutorialKemeny.dot -o tutorialKemeny.png

```

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 performance criteria consensus.

Listing 8.11 Computing the consensus quality of a ranking

```

1 >>> g.showRankingConsensusQuality(ke.maximalRankings[0])
2   Consensus quality of ranking:

```

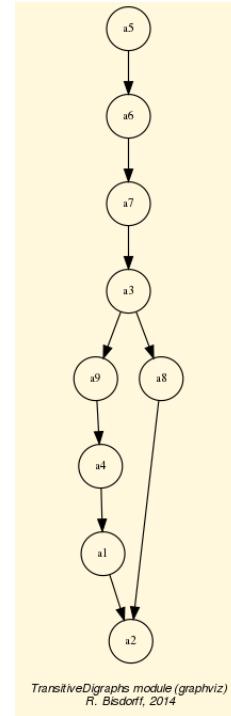


Fig. 8.3 Epistemic disjunction of optimal KEMENY rankings. It is interesting to notice that both KEMENY rankings only differ in their respective positioning of alternative a_8 ; either before or after alternatives a_9 , a_4 and a_1 .

```

3   ['a5', 'a6', 'a7', 'a3', 'a8', 'a9', 'a4', 'a1', 'a2']
4   criterion (weight): correlation
5   -----
6       b09 (0.050) : +0.361
7       b04 (0.050) : +0.333
8       b08 (0.050) : +0.292
9       b01 (0.050) : +0.264
10      c01 (0.167) : +0.250
11      b03 (0.050) : +0.222
12      b07 (0.050) : +0.194
13      b05 (0.050) : +0.167
14      c02 (0.167) : +0.000
15      b10 (0.050) : +0.000
16      b02 (0.050) : -0.042
17      b06 (0.050) : -0.097
18      c03 (0.167) : -0.167
19  Summary:
20      Weighted mean marginal correlation (a): +0.099
21      Standard deviation (b) : +0.177
22      Ranking fairness (a)-(b) : -0.079
23 >>> g.showRankingConsensusQuality(ke.maximalRankings[1])
24  Consensus quality of ranking:
25  ['a5', 'a6', 'a7', 'a3', 'a9', 'a4', 'a1', 'a8', 'a2']
26  criterion (weight): correlation
27  -----

```

```

28      b09 (0.050)   : +0.306
29      b08 (0.050)   : +0.236
30      c01 (0.167)   : +0.194
31      b07 (0.050)   : +0.194
32      c02 (0.167)   : +0.167
33      b04 (0.050)   : +0.167
34      b03 (0.050)   : +0.167
35      b01 (0.050)   : +0.153
36      b05 (0.050)   : +0.056
37      b02 (0.050)   : +0.014
38      b06 (0.050)   : -0.042
39      c03 (0.167)   : -0.111
40      b10 (0.050)   : -0.111
41  Summary:
42      Weighted mean marginal correlation (a) : +0.099
43      Standard deviation (b)                 : +0.132
44      Ranking fairness (a)-(b)               : -0.033

```

Both KEMENY rankings show the same *weighted mean marginal correlation* (+0.099, see Listing 8.11 Lines 19-22, 42-44) with all thirteen performance criteria. However, the second ranking shows a slightly lower *standard deviation* (+0.132 versus +0.177), resulting in a slightly **fairer** ranking result (-0.033 versus -0.079).

When several rankings with maximal Kemeny index are given, the `KemenyRanking` class constructor instantiates the ranking with *highest* mean marginal correlation and, in case of ties, with *lowest* weighted standard deviation. Here we obtain ranking: ['a5', 'a6', 'a7', 'a3', 'a9', 'a4', 'a1', 'a8', 'a2'] (see Line 4 above).

8.5 SLATER rankings

The SLATER ranking rule is identical to KEMENY's, but it is working, instead, on the *median cut polarised* digraph. SLATER's rule is also *invariant* Under the codual transform and delivers again indifferently on g or gcd the following results:

Listing 8.12 Computing a SLATER ranking

```

1 >>> from linearOrders import SlaterRanking
2 >>> sl = SlaterRanking(gcd,orderLimit=9)
3 >>> sl.slaterRanking
4  ['a5', 'a6', 'a4', 'a1', 'a3', 'a7', 'a8', 'a9', 'a2']
5 >>> corr = gcd.computeOrderCorrelation(sl.slaterRanking)
6 >>> sl.showCorrelation(corr)
7  Correlation indexes:
8      Extended Kendall tau      : +0.676
9      Epistemic determination   :  0.230
10     Bipolar-valued equivalence : +0.156
11 >>> len(sl.maximalRankings)
12     7

```

We notice in Listing 8.12 Line 8 that the first SLATER ranking is a rather good fit (+0.676), slightly better apparently than the NETFLOWS ranking result (+0.638).

However, there are in fact 7 such potentially optimal SLATER rankings (see Line 12). The corresponding epistemic disjunction gives the following partial ordering:

Listing 8.13 Computing the epistemic disjunction of optimal SLATER rankings

```

1 >>> slw = RankingsFusion(sl, sl.maximalRankings)
2 >>> slw.exportGraphViz(fileName='tutorialSlater')
3 *---- exporting a dot file for GraphViz tools -----*
4 Exporting to tutorialSlater.dot
5 dot -Grankdir=TB -Tpng tutorialSlater.dot -o tutorialSlater.png

```

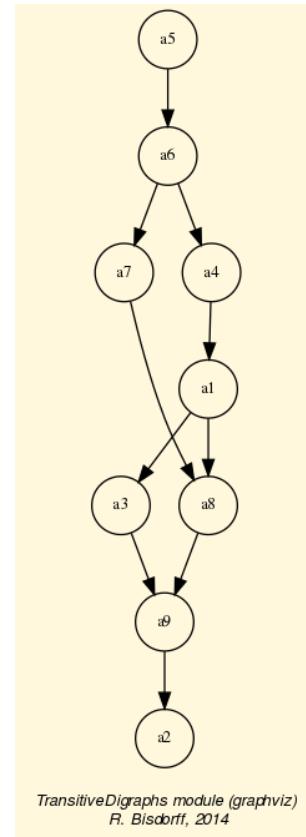


Fig. 8.4 Epistemic disjunction of optimal SLATER rankings. What precise SLATER ranking result should we hence adopt?

KEMENY's and SLATER's ranking rules are furthermore computationally *difficult* problems and effective ranking results are only computable for tiny outranking digraphs (< 20 objects).

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

8.6 KOHLER's ranking-by-choosing rule

KOHLER's *ranking-by-choosing* rule can be formulated like this.

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) 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 8.14 Computing a KOHLER ranking

```

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

```

With this *min-max* lexicographic ranking-by-choosing strategy, we find a correlation result (+0.747) that is until now clearly the nearest to an optimal KEMENY ranking (see Listing 8.9). Only two adjacent pairs: $[a6, a7]$ and $[a8, a9]$ are actually inverted here. Notice that KOHLER's ranking rule, contrary to the previously mentioned rules, is **not** invariant under the codual transform and requires to work on the strict outranking digraph gcd for a better correlation result.

```

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

```

But KOHLER's ranking has a *dual* version, the prudent **Arrow-Raynaud** ordering-by-choosing rule, where a corresponding *max-min* strategy, when used on the *non-strict* outranking digraph g , for ordering the from *last* to *first* produces a similar ranking result (see [LAM-2009], [DIA-2010]).

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

For a ranking (resp. an ordering) result, 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) 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.

Listing 8.15 Ordering-by-choosing with iterated minimal NETFLOWS scores

```

1 >>> from linearOrders import IteratedNetFlowsRanking
2 >>> inf = IteratedNetFlowsRanking(g)
3 >>> inf
4 *----- Digraph instance description -----*
5 Instance class      : IteratedNetFlowsRanking
6 Instance name       : rel_randomCBperftab_ranked
7 Digraph Order       : 9
8 Digraph Size        : 36
9 Valuation domain   : [-1.00;1.00]
10 Determinateness (%) : 100.00
11 Attributes         : ['valuedRanks', 'valuedOrdering',
12                               'iteratedNetFlowsRanking',
13                               'iteratedNetFlowsOrdering',
14                               'name', 'actions', 'order',
15                               'valuationdomain', 'relation',
16                               'gamma', 'notGamma']
17 >>> inf.iteratedNetFlowsRanking
18 ['a5', 'a7', 'a6', 'a3', 'a4', 'a1', 'a8', 'a9', 'a2']
19 >>> corr =
20           g.computeRankingCorrelation(inf.iteratedNetFlowsRanking)
21 >>> g.showCorrelation(corr)
22 Correlation indexes:
23   Crisp ordinal correlation : +0.743
24   Epistemic determination   : 0.230
25   Bipolar-valued equivalence : +0.171
26 >>> inf.iteratedNetFlowsOrdering
27 ['a2', 'a9', 'a1', 'a4', 'a3', 'a8', 'a7', 'a6', 'a5']
28 >>> corr =
29           g.computeOrderCorrelation(inf.iteratedNetFlowsOrdering)
30 >>> g.showCorrelation(corr)
31 Correlation indexes:
32   Crisp ordinal correlation : +0.751
33   Epistemic determination   : 0.230
34   Bipolar-valued equivalence : +0.173

```

The iterated NETFLOWS ranking and its dual, the iterated NETFLOWS ordering, do not usually deliver both the same result (see Lines 18 and 26 above). With 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 slightly better than KOHLER's original result (+0.747, see Listing 8.14 Line 8).

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 Tideman's RANKEDPAIRS ranking rule

Tideman's ranking-by-choosing heuristic, the RANKEDPAIRS rule, working best this time on the non strict outranking digraph g , is based on a *prudent incremental* construction of linear orders that avoids on the fly any cycling outrankings (see [LAM-2009]). The ranking rule may be formulated as follows:

1. Rank the ordered pairs (x, y) of alternatives in decreasing order of $r(x \succsim y) + r(y \not\succsim x)$;
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.

Listing 8.16 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 optimal KEMENY ranking, as we may verify below.

```

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 version, the **Dias-Lamboray** ordering-by-choosing rule, which produces, when working this time on the codual strict outranking digraph gcd , a similar ranking result (see [LAM-2009], [DIA-2010]).

Besides of not providing a unique linear ranking, the ranking-by-choosing rules, as well as their dual, the ordering-by-choosing rules, are unfortunately not scalable to outranking digraphs of larger orders (> 100). For such bigger outranking digraphs, with several hundred or thousands of alternatives, only the COPELAND and the NETFLOWS ranking-by-scoring rules, with a polynomial complexity of $\mathcal{O}(n^2)$, where n is the order of the outranking digraph, remain in fact computationally tractable.

Chapter 9

Rating by sorting into relative performance quantiles

Abstract To be written.

We apply order statistics for sorting a set X of n potential decision actions, evaluated on m incommensurable performance criteria, into q quantile equivalence classes, based on pairwise outranking characteristics involving the quantile class limits observed on each criterion. Thus we may implement a weak ordering algorithm of complexity $O(nmq)$.

9.1 Quantile sorting on a single criterion

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.

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

As the relation $<$ is the dual of \geq ($\not\geq$), it will be sufficient to check that $x \geq m_k$ as well as $x \not\geq M_k$ are true for x to be considered a member of category K .

Upper-closed categories (in a more mathematical integration style) may as well be considered. In this case it is sufficient to check that $m_k \not\geq x$ 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.

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 $Q_0 = \min(X)$, we may, with the following intervals: $[Q_0; Q_1[$, $[Q_1; Q_2[$, ..., $[Q_{q-1}; \infty[$, hence define a set of q lower-closed sorting categories. And, in the case of upper-closed categories, if $Q_q = \max(X)$, we would 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 Quantiles sorting 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 $r(\succsim)$ we may compute the bipolar-valued characteristic of the assertion: x belongs to upper-closed *q*-tiles class \mathbf{q}^k class, resp. lower-closed class \mathbf{q}_k , as

follows:

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

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

The outranking relation \succsim verifying the coduality principle, $-r(\mathbf{q}(p_{q-1}) \succsim x) = r(\mathbf{q}(p_{q-1}) \prec x)$, resp. $r(x \succsim \mathbf{q}(p_q)) = r(x \prec \mathbf{q}(p_q))$.

We may compute, for instance, a five-tiling of a given random performance tableau with the help of the QuantilesSortingDigraph class.

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        : 0.000300
27    Compute profiles : 0.000075
28    Compute relation : 0.02581
29    Weak Ordering     : 0.000052
30
31 >>> qs.showCriteriaQuantileLimits()
32     Quantile Class Limits (q = 5)
33     Upper-closed classes
34     crit.      0.20      0.40      0.60      0.80      1.00
35     -----
36     g1          31.35    41.09    58.53    71.91    98.08
37     g2          27.81    39.19    49.87    61.66    96.18
38     g3          25.10    34.78    49.45    63.97    92.59
39     g4          24.61    37.91    53.91    71.02    89.84
40     g5          26.94    36.43    52.16    72.52    96.25
41     g6          23.94    44.06    54.92    67.34    95.97

```

```

42      g7       30.94   47.40   55.46   69.04   97.10
43
44 >>> qs.showSorting()
45     *--- Sorting results in descending order ---*
46     ]0.80 - 1.00]: ['a22']
47     ]0.60 - 0.80]: ['a03','a07','a08','a11','a14','a17',
48                 'a19','a20','a29','a32','a33','a37',
49                 'a39','a41','a42','a49']
50     ]0.40 - 0.60]: ['a01','a02','a04','a05','a06','a08',
51                 'a09','a16','a17','a18','a19','a21',
52                 'a24','a27','a28','a30','a31','a35',
53                 'a36','a40','a43','a46','a47','a48',
54                 'a49','a50']
55     ]0.20 - 0.40]: ['a04','a10','a12','a13','a15','a23',
56                 'a25','a26','a34','a38','a43','a44',
57                 'a45','a49']
58     ] < - 0.20]: ['a44']

```

Most of the decision actions (26) are gathered in the median quintile $]0.40 - 0.60]$ class, whereas the highest quintile $]0.80 - 1.00]$ and the lowest quintile $] < - 0.20]$ classes gather each one a unique decision alternative ($a22$, resp. $a44$) (see Listing 9.1 Lines 45-).

We may inspect as follows the details of the corresponding sorting characteristics.

Listing 9.2 Bipolar-valued sorting characteristics (extract)

```

1 >>> qs.valuationdomain
2 {'min': Decimal('-1.0'), 'med': Decimal('0'),
3  'max': Decimal('1.0')}
4 >>> qs.showSortingCharacteristics()
5     x in q^k          r(q^k-1 < x)  r(q^k >= x)  r(x in q^k)
6     a22 in ]< - 0.20]    1.00        -0.86        -0.86
7     a22 in ]0.20 - 0.40]    0.86        -0.71        -0.71
8     a22 in ]0.40 - 0.60]    0.71        -0.71        -0.71
9     a22 in ]0.60 - 0.80]    0.71        -0.14        -0.14
10    a22 in ]0.80 - 1.00]    0.14        1.00        0.14
11    ...
12    ...
13    a44 in ]< - 0.20]    1.00        0.00        0.00
14    a44 in ]0.20 - 0.40]    0.00        0.57        0.00
15    a44 in ]0.40 - 0.60]   -0.57        0.86       -0.57
16    a44 in ]0.60 - 0.80]   -0.86        0.86       -0.86
17    a44 in ]0.80 - 1.00]   -0.86        0.86       -0.86
18    ...
19    ...
20    a49 in ]< - 0.20]    1.00        -0.43       -0.43
21    a49 in ]0.20 - 0.40]    0.43        0.00        0.00
22    a49 in ]0.40 - 0.60]    0.00        0.00        0.00
23    a49 in ]0.60 - 0.80]    0.00        0.57        0.00
24    a49 in ]0.80 - 1.00]   -0.57        0.86       -0.57

```

Alternative $a22$ verifies indeed positively both sorting conditions only for the highest quintile $[0.80 - 1.00]$ class (see Listing 9.2 Lines 10). Whereas alternatives

$a44$ and $a49$, for instance, weakly verify both sorting conditions each one for two, resp. three, adjacent quintile classes (see Lines 13-14 and 21-23).

Quantiles sorting results indeed always verify the following **Properties**.

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:** If $r(x \in \mathbf{q}^k) \neq 0$ for $k = 1, \dots, q$, then 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 9.3 Weakly ranking the quintiles sorting result

```

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

```

Following, for instance, the *average* ranking strategy, we find confirmed in the weak ranking shown in Listing 9.3, that alternative $a49$ is indeed sorted into three adjacent quintiles classes, namely $]0.20 - 0.80]$ (see Line 6) and precedes the $]0.40 - 0.60]$ class, of same average of lower and upper limits.

The `QuantilesSortingDigraph` constructor gives 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

We may notice that a given outranking digraph -the association of a set of decision alternatives and an outranking relation- is, following the methodological requirements of the outranking approach, necessarily associated with a corresponding performance tableau. And, we may use this underlying performance tableau for linearly decomposing the set of potential decision alternatives into ordered quantiles equivalence classes by using the quantiles sorting technique seen in the previous Section.

In the coding example shown in Listing 9.4 below, we generate for instance, first (Lines 2-3), a simple performance tableau of 75 decision alternatives and, secondly (Lines 4-6), we construct the corresponding PreRankedOutrankingDigraph instance called *prg*. Notice by the way the *BigData* flag (Line 3) used here for generating a parsimoniously commented performance tableau.

Listing 9.4 Computing a *pre-ranked* sparse outranking digraph

```

1 >>> from randomPerfTabs import RandomPerformanceTableau
2 >>> tp = RandomPerformanceTableau(numberOfActions=75,\n3 ...                                         BigData=True, seed=100)
4 >>> from sparseOutrankingDigraphs import\
5 ...                                         PreRankedOutrankingDigraph
6 >>> prg = PreRankedOutrankingDigraph(Tp, quantiles=5)
7 >>> prg
8     *----- Object instance description -----*
9     Instance class      : PreRankedOutrankingDigraph
10    Instance name       : randomperftab_pr
11    Actions             : 75
12    Criteria            : 7
13    Sorting by          : 5-Tiling
14    Ordering strategy   : average
15    Components          : 9
16    Minimal order       : 1
17    Maximal order        : 25
18    Average order        : 8.3
19    fill rate           : 20.432%
20    Attributes          :
21    ['actions','criteria','evaluation','NA','name',
22     'order','runTimes','dimension','sortingParameters',
23     'valuationdomain','profiles','categories','sorting',
24     'decomposition','nbrComponents','components',
25     'fillRate','minimalComponentSize','maximalComponentSize',
26     ... ]

```

The ordering of the 5-tiling result is following the *average* lower and upper quintile limits strategy (see Listing 9.4 Line 14). We obtain here 9 ordered components of minimal order 1 and maximal order 25. The corresponding *pre-ranked decomposition* may be visualized as follows.

Listing 9.5 The quantiles decomposition of a pre-ranked outranking digraph

```

1 >>> prg.showDecomposition()

```

```

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

```

The highest quintile class ($[80\% - 100\%]$) contains decision alternatives 5, 42, 43 and 47. Lowest quintile class ($[-20\%]$) gathers alternatives 57 and 59 (see Listing 9.5 Lines 3 and 15). We may inspect the resulting sparse outranking relation map as follows in a browser view.

```
1      >>> prg.showHTMLRelationMap()
```

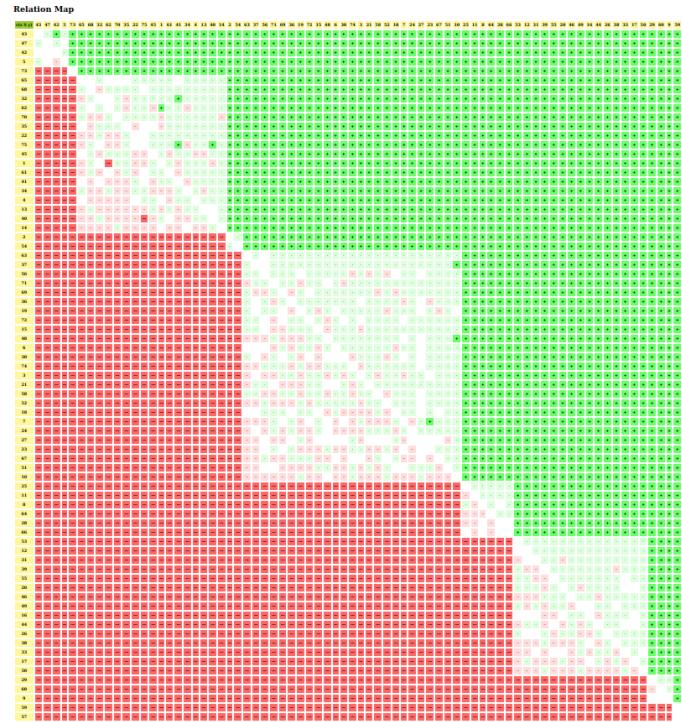


Fig. 9.1 The relation map of a sparse outranking digraph.

In Fig. 9.1 we easily recognize the 9 linearly ordered quantile equivalence classes. *Green* and *light-green* show positive **outranking** situations, whereas positive **outranked** situations are shown in *red* and *light-red*. Indeterminate situations appear in white. In each one of the 9 quantile equivalence classes we recover in fact the corresponding bipolar-valued outranking *sub-relation*, which leads to an actual **fill-rate** of 20.4% (see Listing 9.4 Line 19).

We may now check how faithful the sparse model represents the complete outranking relation.

```

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

```

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

It is worthwhile noticing in Listing 9.4 Lines 20- 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.

```

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

```

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. Otherwise we just check the respective ranks of the components.

9.4 Ranking pre-ranked sparse outranking digraphs

Each one of these 9 ordered components may now be locally ranked by using a suitable ranking rule. Best operational results, both in run times and quality, are more or less equally given with the COPELAND and the NETFLOWS rules. The eventually obtained linear ordering (from the worst to best) is stored in a `prg.boostedOrder` attribute. A reversed linear ranking (from the best to the worst) is stored in a `prg.boostedRanking` attribute.

```

1 >>> prg.boostedRanking
2 [43, 47, 42, 5, 73, 65, 68, 32, 62, 70, 35, 22, 75, 45, 1,
3   61, 41, 34, 4, 13, 40, 14, 2, 54, 63, 37, 56, 71, 69, 36,
4   19, 72, 15, 48, 6, 30, 74, 3, 21, 58, 52, 18, 7, 24, 27,
5   23, 67, 51, 10, 25, 11, 8, 64, 28, 66, 53, 12, 31, 39, 55,
6   20, 46, 49, 16, 44, 26, 38, 33, 17, 50, 29, 60, 9, 59, 57]
```

Alternative 43 appears *first ranked*, whereas alternative 57 is *last ranked* (see Line 2 and 6 above). The quality of this ranking result may be assessed by computing its ordinal correlation with the standard outranking relation.

```

1 >>> corr = g.computeRankingCorrelation(prg.boostedRanking)
2 >>> g.showCorrelation(corr)
3 Correlation indexes:
4   Crisp ordinal correlation : +0.807
5   Epistemic determination   : 0.315
6   Bipolar-valued equivalence : +0.254
```

We may also verify below that the COPELAND ranking obtained from the standard outranking digraph is highly correlated (+0.822) with the one obtained from the sparse outranking digraph.

```

1 >>> from linearOrders import CopelandOrder
2 >>> cop = CopelandOrder(g)
3 >>> print(cop.computeRankingCorrelation(prg.boostedRanking))
4   {'correlation': 0.822, 'determination': 1.0}
```

Noticing the computational efficiency of the quantiles sorting construction, coupled with the separability property of the quantile class membership characteristics computation, we will make usage of the `PreRankedOutrankingDigraph` constructor for HPC ranking big and even huge performance tableaux (see Chapter 11).

Chapter 10

Rating by ranking with learned performance quantile norms

Abstract To be written.

10.1 The absolute rating problem

In this tutorial we address the problem of rating multiple criteria performances of a set of potential decision alternatives with respect to empirical order statistics, i.e. performance quantiles learned from historical performance data gathered from similar decision alternatives observed in the past (see [CPSTAT-L5]).

To illustrate the decision problem we face, consider for a moment that, in a given decision aid study, we observe, for instance in the Table 10.1 below, the multi-criteria performances of two potential decision alternatives, named $a1001$ and $a1010$, marked on 7 *incommensurable* preformance criteria: 2 *Costs* criteria $c1$ and $c2$ (to minimize) and 5 *Benefits* criteria $b1$ to $b5$ (to maximize).

Table 10.1 Multi-criteria performances of two potential decision alternatives

Criterion (weight)	$b1$ (2)	$b2$ (2)	$b3$ (2)	$b4$ (2)	$b5$ (b2)	$c1$ (5)	$c2$ (5)
$a1001$	37.0	2	2	61.0	31.0	-4	-40.0
$a1010$	32.0	9	6	55.0	51.0	-4	-35.0

The performances on *Benefits* criteria $b1$, $b4$ and $b5$ are measured on a cardinal scale from 0.0 (worst) to 100.0 (best) whereas, the performances on the *Benefits* criteria $b2$ and $b3$ and on the *Costs* criterion $c1$ are measured on an ordinal scale from 0 (worst) to 10 (best), respectively -10 (worst) to 0 (best). The performances on the *Costs* criterion $c2$ are again measured on a cardinal negative scale from -100.00 (worst) to 0.0 (best).

The importance (sum of weights) of the *Costs* criteria is **equal** to the importance (sum of weights) of the *Benefits* criteria taken all together.

The non trivial decision problem we now face here, is to decide, how the multiple criteria performances of $a1001$, respectively $a1010$, may be rated (*excellent?* *good?*, or *fair?*; perhaps even, *weak?* or *very weak?*) in an **order statistical sense**, when compared with all potential similar multi-criteria performances one has already encountered in the past.

To solve this *absolute* rating decision problem, first, we need to estimate multi-criteria **performance quantiles** from historical records.

10.2 Incremental learning of historical performance quantiles

Suppose that we see flying in random multiple criteria performances from a given model of random performance tableau (see Chapter 5). The question we address here is to estimate empirical performance quantiles on the basis of so far observed performance vectors. For this task, we are inspired by [CHAM-2006] and [NR3-2007], who present an efficient algorithm for incrementally updating a quantile-binned cumulative distribution function (CDF) with newly observed CDFs.

The `PerformanceQuantiles` class implements such a performance quantiles estimation based on a given performance tableau. Its main attributes are:

- Ordered `objectives` and `criteria` dictionaries from a valid performance tableau instance;
- A list `quantileFrequencies` of quantile frequencies like:
 - `quartiles` $[0.0, 0.25, 0.5, 0.75, 1.0]$,
 - `quintiles` $[0.0, 0.2, 0.4, 0.6, 0.8, 1.0]$ or
 - `deciles` $[0.0, 0.1, 0.2, \dots, 1.0]$ for instance;
- An ordered dictionary `limitingQuantiles` of so far estimated *lower* (default) or *upper* quantile class limits for each frequency per criterion;
- An ordered dictionary `historySizes` for keeping track of the number of evaluations seen so far per criterion. Missing data may make these sizes vary from criterion to criterion.

Below, an example Python session concerning 900 decision alternatives randomly generated from a *Cost-Benefit* Performance tableau model (see Section 5.3) from which are also drawn the performances of alternatives $a1001$ and $a1010$ shown in Table 10.1 above.

Listing 10.1 Computing performance quantiles from a given performance tableau

```

1 >>> from performanceQuantiles import PerformanceQuantiles
2 >>> from randomPerfTabs import RandomCBPerformanceTableau
3 >>> nbrActions=900
4 >>> nbrCrit = 7
5 >>> seed = 100

```

```

6 >>> tp = RandomCBPerformanceTableau(numberOfActions=nbrActions,\n7 ...                               numberOfCriteria=nbrCrit, seed=seed)\n8\n9 >>> pq = PerformanceQuantiles(tp,\n10 ...                                numberOfBins = 'quartiles',\n11 ...                                LowerClosed=True)\n12\n13 >>> pq\n14 *----- PerformanceQuantiles instance description -----*\n15     Instance class      : PerformanceQuantiles\n16     Instance name       : 4-tiled_performances\n17     Objectives          : 2\n18     Criteria            : 7\n19     Quantiles           : 4\n20     History sizes      : {'c1': 887, 'b1': 888, 'b2': 891, 'b3': 895,\n21 ...                           'b4': 892, 'c2': 893, 'b5': 887}\n22     Attributes          : ['perfTabType', 'valueDigits',\n23 ...                           'actionsTypeStatistics',\n24 ...                           'objectives', 'BigData',\n25 ...                           'missingDataProbability',\n26 ...                           'criteria', 'LowerClosed', 'name',\n27 ...                           'quantilesFrequencies', 'historySizes',\n28 ...                           'limitingQuantiles', 'cdf']\n
```

The `PerformanceQuantiles` class parameter `numberOfBins` (see Listing 10.1 Line 10 above), choosing 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*.

Listing 10.2 Printing out the estimated quartile limits

```

1 >>> pq.showLimitingQuantiles(ByObjectives=True)\n2     ---- Historical performance quantiles ----*\n3     Costs\n4     criteria | weights | '0.00' '0.25' '0.50' '0.75' '1.00'\n5     -----|-----\n6     'c1'   |    5   | -10     -7     -5     -3     0\n7     'c2'   |    5   | -96.37 -70.65 -50.10 -30.00 -1.43\n8     Benefits\n9     criteria | weights | '0.00' '0.25' '0.50' '0.75' '1.00'\n10    -----|-----\n11    'b1'   |    2   | 1.99    29.82 49.44 70.73 99.83\n12    'b2'   |    2   | 0        3       5       7       10\n13    'b3'   |    2   | 0        3       5       7       10\n14    'b4'   |    2   | 3.27    30.10 50.82 70.89 98.05\n15    'b5'   |    2   | 0.85    29.08 48.55 69.98 97.56\n
```

Both objectives are *equi-important*; the sum of weights (10) of the *Costs* criteria balance the sum of weights (10) of the *Benefits* criteria (see Listing 10.2 column 2). The preference direction of the *Costs* criteria *c1* and *c2* is *negative*; the lesser the costs the better it is, whereas all the *Benefits* criteria *b1* to *b5* show *positive* preference directions, i.e. the higher the benefits the better it is. The columns entitled '0.00', resp. '1.00' show the *quartile Q0*, resp. *Q4*, i.e. the *worst*, resp. *best* per-

formance observed so far on each criterion. Column '0.50' shows the *median* (Q_2) performance observed on the criteria.

New decision alternatives with random multiple criteria performance vectors from the same random performance tableau model may now be generated with ad hoc random performance generators. We provide for experimental purpose, in the `randomPerfTabs` module, three such generators:

- One for the standard `RandomPerformanceTableau` model (see Section 5.2),
- One for the two objectives `RandomCBPerformanceTableau` Cost-Benefit model (see Section 5.3), and
- One for the `Random3ObjectivesPerformanceTableau` model with three objectives concerning respectively *economic*, *environmental* or *social* aspects (see Section 5.4).

Given a new Performance Tableau with 100 new decision alternatives, the so far estimated historical quantile limits may be updated as follows:

Listing 10.3 Generating 100 new random decision alternatives of the same model

```

1 >>> from randomPerfTabs import RandomPerformanceGenerator
2 >>> rpg = RandomPerformanceGenerator(tp, seed=seed)
3 >>> newTab = rpg.randomPerformanceTableau(100)
4 >>> # Updating the quartile norms shown above
5 >>> pq.updateQuantiles(newTab, historySize=None)
```

Parameter `historySize` (see Listing 10.3 Line 5) of the `updateQuantiles()` method allows to *balance* the *new* evaluations against the *historical* ones. With `historySize = None` (the default setting), the balance in the example above is 900/1000 (90%, weight of historical data) against 100/1000 (10%, weight of the new incoming observations). Putting `historySize = 0`, for instance, will ignore all historical data (0/100 against 100/100) and restart building the quantile estimation with solely the new incoming data. The updated quantile limits may be shown in a browser view (see Fig. 10.1).

```

1 >>> # showing the updated quantile limits in a browser view
2 >>> pq.showHTMLLimitingQuantiles(Transposed=True)
```

10.3 Rating-by-ranking new performances with quantile norms

For *rating* a newly given set of decision alternatives with the help of empirical performance quantiles estimated from historical data, we provide the `LearnedQuantilesRatingDigraph` class, a specialisation of the `QuantilesSortingDigraph` class (see Chapter 9). The absolute rating result is computed by *ranking* the new performance records together with the learned quantile limits. The constructor requires a valid `PerformanceQuantiles` instance. By default, the constructor uses COPELAND's or the NETFLOWS ranking rule whichever fits best with the underlying outranking digraph.

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 the updated quartiles limits.

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. The `actions` attribute in such a `LearnedQuantilesRatingDigraph` class instance contain not only the newly given decision alternatives, but also the historical quantile profiles obtained from a given `PerformanceQuantiles` class instance, i.e. estimated quantile bins' performance limits from historical performance data.

We reconsider the `PerformanceQuantiles` object instance `pq` as computed in the previous section. Let `newActions` be a list of 10 new decision alternatives generated with the same random performance tableau model and including the two decision alternatives `a1001` and `a1010` mentioned at the beginning.

Listing 10.4 Computing the absolute rating of 10 new decision alternatives

```

1 >>> from sortingDigraphs import LearnedQuantilesRatingDigraph
2 >>> newActions = rpg.randomActions(10)
3 >>> lqr = LearnedQuantilesRatingDigraph(pq,newActions,\n4 ...                                         rankingRule='best')
5 >>> lqr
6     ----- Object instance description
7     Instance class      : LearnedQuantilesRatingDigraph
8     Instance name       : normedRatingDigraph
9     Criteria            : 7
10    Quantile profiles   : 4
11    Lower-closed bins   : True
12    New actions          : 10
13    Size                 : 93
14    Determinateness (%) : 76.1
15    Ranking rule         : Copeland
16    Ordinal correlation : +0.95
17    Attributes: ['runTimes','objectives','criteria',

```

```

18     'LowerClosed','quantilesFrequencies',
19     'limitingQuantiles','historySizes','cdf','name',
20     'newActions','evaluation','categories',
21     'criteriaCategoryLimits','profiles','profileLimits',
22     'hasNoVeto','actions','completeRelation','relation',
23     'concordanceRelation','valuationdomain','order',
24     'gamma','notGamma','rankingRule','rankingCorrelation',
25     'rankingScores','actionsRanking','ratingCategories',
26     'ratingRelation','relationOrig']
27 *---- Constructor run times (in sec.)
28     Threads          : 1
29     Total time       : 0.02218
30     Data input       : 0.00134
31     Quantile classes: 0.00008
32     Compute profiles: 0.00021
33     Compute relation: 0.01869
34     Compute rating   : 0.00186
35     Compute sorting  : 0.00000

```

Data input to the `LearnedQuantilesRatingDigraph` class constructor (see Listing 10.4 Line 3) are a valid `PerformanceQuantiles` object *pq* and a compatible list `newActions` of new decision alternatives generated from the same random generator *rpg* (see Listing 10.3 Line 2).

Let us have a look at the digraph's nodes, here called `newActions`.

Listing 10.5 Performance tableau of the new incoming decision alternatives

```

1 >>> lqr.showPerformanceTableau(actionsSubset=lqr.newActions)
2 *---- performance tableau ----*
3 criteria | a1001 a1002 a1003 a1004 a1005 a1006 a1007 a1008 a1009 a1010
4 -----
5   'b1' | 37.0 27.0 24.0 16.0 42.0 33.0 39.0 64.0 42.0 32.0
6   'b2' | 2.0 5.0 8.0 3.0 3.0 3.0 6.0 5.0 4.0 9.0
7   'b3' | 2.0 4.0 2.0 1.0 6.0 3.0 2.0 6.0 6.0 6.0
8   'b4' | 61.0 54.0 74.0 25.0 28.0 20.0 20.0 49.0 44.0 55.0
9   'b5' | 31.0 63.0 61.0 48.0 30.0 39.0 16.0 96.0 57.0 51.0
10  'c1' | -4.0 -6.0 -8.0 -5.0 -1.0 -5.0 -1.0 -6.0 -6.0 -4.0
11  'c2' | -40.0 -23.0 -37.0 -37.0 -24.0 -27.0 -73.0 -43.0 -94.0 -35.0

```

Among the 10 new incoming decision alternatives (see Listing 10.5), we recognize alternatives *a1001* (see column 2) and *a1010* (see last column) we have shown in Table 10.1.

The `LearnedQuantilesRatingDigraph` class instance's `actions` dictionary includes as well the closed lower limits of the four quartile classes: *m1* = [0.0 – [, *m2* = [0.25 – [, *m3* = [0.5 – [, *m4* = [0.75 – [. We find these limits in a `profiles` attribute (see Listing 10.6 below).

Listing 10.6 Showing the limiting profiles of the rating quantiles

```

1 >>> lqr.showPerformanceTableau(actionsSubset=lqr.profiles)
2 *---- Quartiles limit profiles ----*
3 criteria | 'm1'   'm2'   'm3'   'm4'
4 -----
5   'b1' | 2.0    28.8   49.6   75.3
6   'b2' | 0.0    2.9    4.9    6.7
7   'b3' | 0.0    2.9    4.9    8.0

```

```

8      'b4' | 3.3   35.9   58.6   72.0
9      'b5' | 0.8   32.8   48.1   69.7
10     'c1' | -10.0 -7.4   -5.4   -3.4
11     'c2' | -96.4 -72.2  -52.3  -34.0

```

The main run time (see Listing 10.4 Lines 23-29) is spent by the class constructor in computing a bipolar-valued outranking relation on the extended actions set including both the new alternatives as well as the quartile class limits. In case of large volumes, i.e. many new decision alternatives and centile classes for instance, a multi-threading version may be used when multiple processing cores are available.

The actual rating procedure will rely on a complete ranking of the new decision alternatives as well as the quartile class limits obtained from the corresponding bipolar-valued outranking digraph. Two efficient and scalable ranking rules, the COPELAND and its valued version, the NETFLOWS rule may be used for this purpose. The `rankingRule` parameter allows to choose one of both. With `rankingRule='best'` (see Listing 10.4 Line 3) the `LearnedQuantilesRatingDigraph` constructor will choose the ranking rule that results in the highest ordinal correlation with the given outranking relation (see Chapter 22 and [BIS-2012]).

In this rating example, the COPELAND rule appears to be the more appropriate ranking rule.

Listing 10.7 COPELAND ranking of new alternatives and historical quartile limits

```

1 >>> lqr.rankingRule
2     'Copeland'
3 >>> lqr.actionsRanking
4     ['m4', 'a1005', 'a1010', 'a1002', 'a1008', 'a1006', 'a1001',
5      'a1003', 'm3', 'a1007', 'a1004', 'a1009', 'm2', 'm1']
6 >>> lqr.showCorrelation(lqr.rankingCorrelation)
7     Correlation indexes:
8         Crisp ordinal correlation : +0.945
9         Epistemic determination : 0.522
10        Bipolar-valued equivalence : +0.493

```

We achieve here (see Listing 10.7) 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 m_1 to m_4 , which is very close in an ordinal sense ($\tau = 0.945$) to the underlying strict outranking relation.

The eventual rating procedure is based in this example on the *lower* quartile limits, such that we may collect the quartile classes' contents in increasing order of the *quartiles*.

```

1 >>> lqr.ratingCategories
2 OrderedDict([
3     ('m2', ['a1007', 'a1004', 'a1009']),
4     ('m3',
5      ['a1005', 'a1010', 'a1002', 'a1008', 'a1006', 'a1001', 'a1003'])
6 ])

```

We notice above that no new decision alternatives are actually rated in the lowest $[0.0 - 0.25]$, respectively highest $[0.75 -]$ quartile classes. Indeed, the rating result is shown, in descending order, as follows:

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 more conveniently be consulted in a browser view via a specialised rating heatmap format:

```

1 >>> lqr.showHTMLRatingHeatmap(
2 ...             pageTitle='Heatmap of Quartiles Rating', \
3 ...             Correlations=True, colorLevels=5)

```

Heatmap of Quartiles Rating

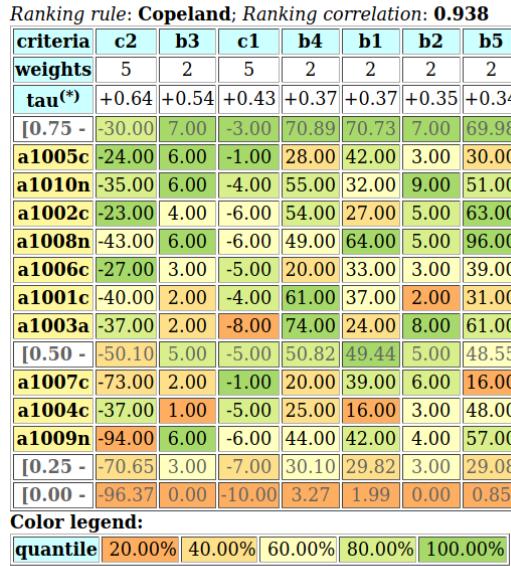


Fig. 10.2 Heatmap of absolute quartiles ranking.

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

Using furthermore a specialised version of the `exportGraphViz()` method allows drawing the same rating result in a Hasse diagram format (see Fig. 10.3).

```

1 >>> lqr.exportRatingGraphViz('normedRatingDigraph')
2     ----- exporting a dot file for GraphViz tools -----
3     Exporting to normedRatingDigraph.dot
4     dot -Grankdir=TB -Tpng normedRatingDigraph.dot -o
          normedRatingDigraph.png

```

We may now answer the *absolute rating* decision problem stated at the beginning. Decision alternative $a1001$ and alternative $a1010$ (see Table 10.1) are both rated into

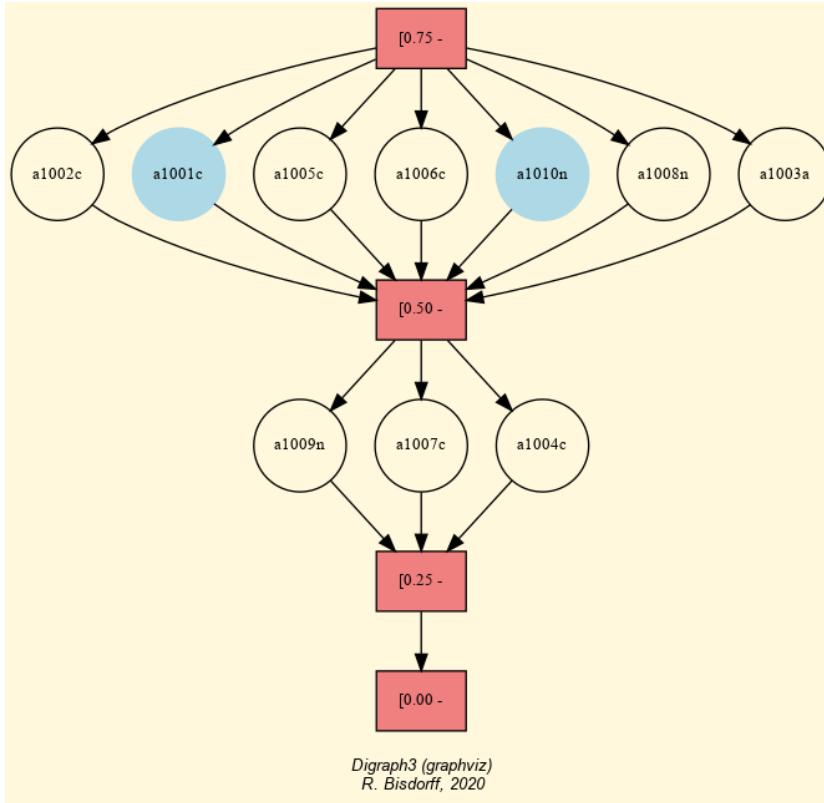


Fig. 10.3 Absolute quartiles rating digraph

the same third quartile $Q3$ class (see Fig. 10.3), even if the COPELAND ranking, obtained from the underlying strict outranking digraph (see 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(tp, numberOfBins = 'deciles', \
2 ...                                LowerClosed=True)
3 >>> pq1.updateQuantiles(newTab, historySize=None)
4 >>> lqr1 =
5     LearnedQuantilesRatingDigraph(pq1,newActions,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 $[0.50 - 0.75[$, 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 $a1009$), alternatives $a1004$ and $a1007$ are now rated into the 5th decile class $[0.40 - 0.50[$ and $a1009$ is lowest rated into the 4th decile class $[0.30 - 0.40[$.

A browser view may again more conveniently illustrate this refined rating result (see Fig. 10.4).

```
1 >>> lqr1.showHTMLRatingHeatmap(\n2 ...     pageTitle='Heatmap of the deciles rating', \n3 ...     colorLevels=5, Correlations=True)
```

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.

Fig. 10.4 Heatmap of absolute deciles rating

In this *deciles* rating, decision alternatives $a1001$ and $a1010$ are now, as expected, rated in the 6th decile ($D6$), respectively in the 7th decile ($D7$).

To avoid having to recompute performance deciles from historical data when wishing to refine a rating result, it is useful, depending on the actual size of the historical data, to initially compute performance quantiles with a relatively high number of bins, for instance *dodeciles* or even *centiles*. It is then possible to correctly interpolate *quartiles* or *deciles* for instance, when constructing the rating digraph.

Listing 10.10 From deciles interpolated quartiles rating result

```

1 >>> lqr2 = LearnedQuantilesRatingDigraph(pql,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']]
```

With the *quantiles* parameter (see Listing 10.10 Line 2), we may recover by interpolation the same quartiles rating as obtained directly with historical performance quartiles (see Listing 10.8). Mind that a correct interpolation of quantiles from a given cumulative distribution function requires more or less uniform distributions of observations in each bin.

More generally, in the case of industrial production monitoring problems, for instance, where large volumes of historical performance data may be available, it may be of interest to estimate even more precisely the marginal cumulative distribution functions, especially when *tail* rating results, i.e. distinguishing *very 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.

Chapter 11

HPC ranking of big performance tableaux

Abstract To be written.

11.1 C-compiled Python modules

The Digraph3 collection provides cythonized Footnote[6], i.e. C-compiled and optimised versions of the main python modules for tackling multiple criteria decision problems facing very large sets of decision alternatives (> 10000). Such problems appear usually with a combinatorial organisation of the potential decision alternatives, as is frequently the case in bioinformatics for instance. If HPC facilities with nodes supporting numerous cores (> 20) and big RAM ($> 50\text{GB}$) are available, ranking up to several millions of alternatives (see [BIS-2016]) becomes effectively tractable.

Four cythonized DIGRAPH3 modules, prefixed with the letter `c` and taking a `.pyx` extension, are provided with their corresponding setup tools in the `cython` directory of the DIGRAPH3 resources, namely:

```
cRandPerfTabs.pyx,  
cIntegerOutrankingDigraphs.pyx,  
cIntegerSortingDigraphs.pyx, and  
cSparseIntegerOutrankingDigraphs.pyx.
```

Their automatic compilation and installation, alongside the standard DIGRAPH3 python3 modules, requires the `cython` compiler Footnote[6] (`...$ python3 -m pip install cython`) and a C compiler (`...$ sudo apt install gcc` on Ubuntu).

These cythonized modules, specifically designed for being run on HPC clusters (see <https://hpc.uni.lu>), require the Unix *forking* start method of subprocesses (see start methods of the [multiprocessing library](#)) and therefore, due

to forking problems on Mac OS platforms, may only operate safely on Linux platforms.

11.2 Big Data performance tableaux

In order to efficiently type the C variables, the `cRandPerfTabs` module provides the usual random performance tableau models, 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.

Listing 11.1 Big data performance tableau format

```

1 >>> from cRandPerfTabs import cRandomPerformanceTableau
2 >>> t = cRandomPerformanceTableau(numberOfActions=4,\n3 ...                                numberOfCriteria=2)
4 >>> t
5      *----- PerformanceTableau instance description -----*
6      Instance class      : cRandomPerformanceTableau
7      Seed                 : None
8      Instance name        : cRandomperftab
9      Actions              : 4
10     Criteria             : 2
11     Attributes           : ['randomSeed', 'name', 'actions',
12                           'criteria', 'evaluation',
13                           'weightPreorder']
14 >>> t.actions
15     OrderedDict([(1, {'name': '#1'}), (2, {'name': '#2'}),
16                  (3, {'name': '#3'}), (4, {'name': '#4'})])
17 >>> t.criteria
18     OrderedDict([
19       ('g1', {'name': 'RandomPerformanceTableau() instance',
20               'thresholds': {'ind': (10.0, 0.0),
21                             'pref': (20.0, 0.0),
22                             'veto': (80.0, 0.0)},
23               'scale': (0.0, 100.0),
24               'weight': 1,
25               'preferenceDirection': 'max'}),
26       ('g2', {'name': 'RandomPerformanceTableau() instance',
27               'thresholds': {'ind': (10.0, 0.0),
28                             'pref': (20.0, 0.0),
29                             'veto': (80.0, 0.0)},
30               'scale': (0.0, 100.0),
31               'weight': 1,
32               'preferenceDirection': 'max'}))]
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 | 'q1'    'q2'
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

```

Conversions from the Big Data model to the standard model and vice versa are provided.

```

1 >>> t1 = t.convert2Standard()
2 >>> t1.convertWeight2Decimal()
3 >>> t1.convertEvaluation2Decimal()
4 >>> t1
5     *----- PerformanceTableau instance description -----
6     Instance class : PerformanceTableau
7     Seed          : None
8     Instance name : std_cRandomperftab
9     Actions       : 4
10    Criteria      : 2
11    Attributes    : ['name', 'actions', 'criteria',
12                      'weightPreorder', 'evaluation',
13                      'randomSeed']

```

11.3 C-implemented integer-valued outranking digraphs

The C compiled version of the bipolar-valued digraph model takes integer relation characteristic values.

Listing 11.2 Constructing big outranking digraphs

```

1 >>> t = cRandomPerformanceTableau(numberOfActions=1000,\n2 ...                               numberOfCriteria=2)
3 >>> from cIntegerOutrankingDigraphs import\
4 ...                               IntegerBipolarOutrankingDigraph
5 >>> g = IntegerBipolarOutrankingDigraph(t,\n6 ...                               Threading=True, nbrCores=4)
7 >>> g
8     *----- Object instance description -----
9     Instance class : IntegerBipolarOutrankingDigraph
10    Instance name  : rel_cRandomperftab
11    Actions        : 1000
12    Criteria       : 2
13    Size           : 465024
14    Determinateness : 56.877
15    Valuation domain : {'min': -2, 'med': 0, 'max': 2,
16                          'hasIntegerValuation': True}
17    ---- Constructor run times (in sec.) ----
18    Total time      : 4.23880

```

```

19 Data input      : 0.01203
20 Compute relation : 3.60788
21 Gamma sets     : 0.61889
22 Threads        : 4
23 Attributes      : ['name', 'actions', 'criteria',
24   'totalWeight', 'valuationdomain',
25   'methodData', 'evaluation',
26   'order', 'runTimes', 'nbrThreads',
27   'relation', 'gamma', 'notGamma']

```

On a classic intel-i7 equipped PC with four single threaded cores, the `IntegerBipolarOutrankingDigraph` constructor takes about four seconds for computing a *million* pairwise outranking characteristic values (see Listing 11.2). In a similar setting, the standard `BipolarOutrankingDigraph` constructor operates more than two times slower.

```

1 >>> from outrankingDigraphs import BipolarOutrankingDigraph
2 >>> t1 = t.convert2Standard()
3 >>> g1 = BipolarOutrankingDigraph(t1,\n4 ...                           Threading=True,nbrCores=4)
5 >>> g1
6 *----- Object instance description -----*
7   Instance class    : BipolarOutrankingDigraph
8   Instance name     : rel_std_cRandomperftab
9   Actions           : 1000
10  Criteria          : 2
11  Size              : 465024
12  Determinateness   : 56.817
13  Valuation domain : {'min': Decimal('-1.0'),
14    'med': Decimal('0.0'),
15    'max': Decimal('1.0'),
16    'precision': Decimal('0')}
17  ---- Constructor run times (in sec.) ----
18  Total time        : 8.63340
19  Data input         : 0.01564
20  Compute relation  : 7.52787
21  Gamma sets        : 1.08987
22  Threads           : 4

```

By far, most of the run time is in each case needed for computing the individual pairwise outranking characteristic values. Notice also below the memory occupations of both outranking digraph instances.

```

1 >>> from digraphsTools import total_size
2 >>> total_size(g)
3 108662777
4 >>> total_size(g1)
5 212679272
6 >>> total_size(g1.relation)/total_size(g1)
7 0.34
8 >>> total_size(g1.gamma)/total_size(g1)
9 0.45

```

About 103MB for `g` and 202MB for `g1`. The standard `Decimal` valued `BipolarOutrankingDigraph` instance `g1` thus nearly doubles the memory

occupation of the corresponding `IntegerBipolarOutrankingDigraph` `g` instance (see Line 3 and 5 above). 3/4 of this memory occupation is due to the `g1.relation` (34%) and the `g1.gamma` (45%) dictionaries. And these ratios quadratically grow with the digraph order. To limit the object sizes for really big outranking digraphs, we need to abandon the complete implementation of adjacency tables and gamma functions.

11.4 The sparse outranking digraph implementation

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)

```

We sort the 100 decision alternatives into overlapping quartile classes and rank with respect to the average quantile limits.

Listing 11.3 Constructing the sparse integer outranking digraph

```

1 >>> from cSparseIntegerOutrankingDigraphs import \
2 ...                               SparseIntegerOutrankingDigraph
3 >>> sg = SparseIntegerOutrankingDigraph(t, quantiles=4)
4 >>> sg
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 $c3$.

```

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 resulting global relation map of the first ranked 75 alternatives looks as follows.

```
1 >>> sg.showRelationMap(toIndex=75)
```

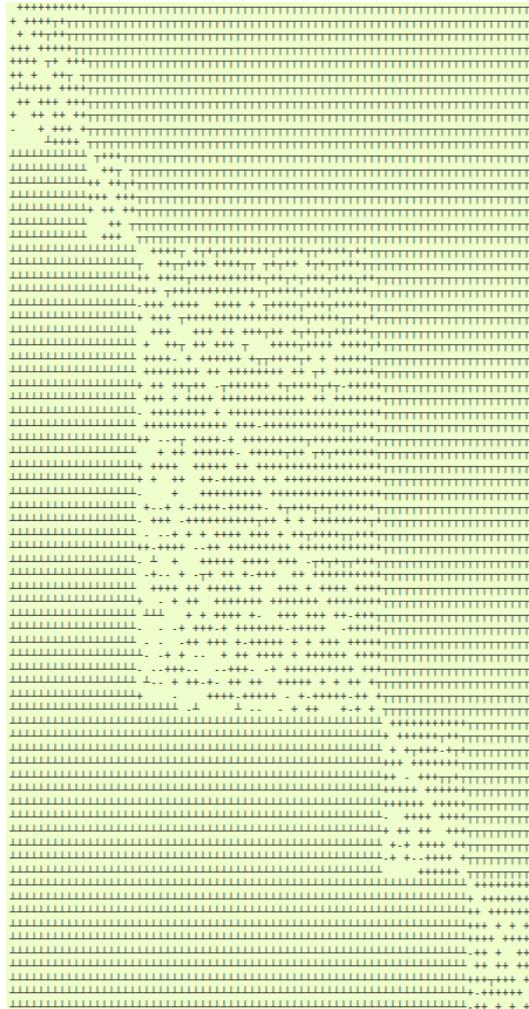


Fig. 11.1 Sparse quartiles-sorting decomposed outranking relation (extract). **Legend:** outranking for certain (\top); outranked for certain (\perp); more or less outranking (+); more or less outranked (−); indeterminate ().

With a fill rate of 25%, the memory occupation of this sparse outranking digraph `sg` instance takes now only 769kB, compared to the 1.7MB required by a corresponding standard `BipolarOutrankingDigraph` instance (see Listing ??).

```
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.

Listing 11.4 The `relation()` function of a sparse outranking digraph

```
1 def relation(self, int x, int y):
2     """
3     *Parameters*:
4         * x (int action key),
5         * y (int action key).
6     Dynamic construction of the global outranking
7     characteristic function *r(x S y)*.
8     """
9     cdef int Min, Med, Max, rx, ry
10    Min = self.valuationdomain['min']
11    Med = self.valuationdomain['med']
12    Max = self.valuationdomain['max']
13    if x == y:
14        return Med
15    cx = self.actions[x]['component']
16    cy = self.actions[y]['component']
17    #print(self.components)
18    rx = self.components[cx]['rank']
19    ry = self.components[cy]['rank']
20    if rx == ry:
21        try:
22            rxpath = self.components[cx]['subGraph'].relation
23            return rxpath[x][y]
24        except AttributeError:
25            componentRanking = self.components[cx]['componentRanking']
26            if componentRanking.index(x) < componentRanking.index(y):
27                return Max
28            else:
29                return Min
30        elif rx > ry:
31            return Min
32        else:
33            return Max
```

11.5 Ranking big performance tableaux

We may now rank the complete set of 100 decision alternatives by locally ranking with the COPELAND or the NETFLOWS rule, for instance, all these individual components.

```
1 >>> sg.boostedRanking
2 [22, 53, 3, 34, 56, 62, 24, 44, 44, 50, 93, 41, 63, 29, 58,
3 96, 7, 43, 81, 91, 35, 25, 76, 66, 65, 8, 10, 1, 11, 61,
4 30, 48, 45, 68, 5, 89, 57, 59, 85, 82, 73, 33, 94, 70,
5 97, 20, 92, 71, 90, 95, 21, 28, 2, 36, 87, 40, 98, 46, 55,
6 100, 64, 17, 26, 27, 19, 69, 6, 38, 4, 37, 60, 31, 77, 78,
```

```

7   47, 99, 18, 12, 80, 54, 88, 39, 9, 72, 86, 42, 13, 23, 67,
8   52, 15, 32, 49, 51, 74, 16, 14, 75, 79, 83, 84]

```

When actually computing linear rankings of a set of alternatives, the local outranking relations are of no practical usage, and we may furthermore reduce the memory occupation of the resulting digraph by

1. refining the ordering of the quantile classes by taking into account how well an alternative is outranking the lower limit of its quantile class, respectively the upper limit of its quantile class is *not* outranking the alternative;
2. dropping the local outranking digraphs and keeping for each quantile class only a locally ranked list of alternatives.

We provide therefore the `cQuantilesRankingDigraph` class.

Listing 11.5 Ranking the sparse integer outranking digraph

```

1 >>> from cSparseIntegerOutrankingDigraphs import \
2 ...           cQuantilesRankingDigraph
3 >>> qr = cQuantilesRankingDigraph(t,4)
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 equivalence classes (see Listing 11.5 Line 13)

```

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,
23             'lowQtileLimit': ']0.25',
24             'highQtileLimit': '0.50'],
25            'componentRanking': [52, 16, 86]}),
26 ('c47', {'rank': 47,
27             'lowQtileLimit': ']<',
28             'highQtileLimit': '0.25'],
29            'componentRanking': [79, 83, 84]})))
30 >>> print('%.0fkB' % (total_size(qr)/1024) )
31 208kB

```

We observe in the Listing above an even more considerably less voluminous memory occupation: 208kB compared to the 769kB 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.

.. code-block:: pycon :linenos:

Listing 11.6 Measuring the loss of quality with respect to the standard outranking digraph

```

1 >>> from cIntegerOutrankingDigraphs import *
2 >>> ig = IntegerBipolarOutrankingDigraph(t)
3 >>> print('Complete outranking : %.4f' \
4 ...   % (ig.computeOrderCorrelation(\n5 ...     ig.computeCopelandOrder()['correlation'])))
6 Complete outranking : +0.7474
7
8 >>> print('Sparse 4-tiling : %.4f' \
9 ...   % (ig.computeOrderCorrelation(\n10 ...     list(reversed(sg.boostedRanking))) \
11 ...     ['correlation'])))
12 Sparse 4-tiling : +0.7172
13
14 >>> print('Optimzed sparse 4-tiling: %.4f' \
15 ...   % (ig.computeOrderCorrelation(\n16 ...     list(reversed(qr.boostedRanking))) \
17 ...     ['correlation'])))
18 Optimzed sparse 4-tiling: +0.7051

```

The best ranking correlation with the pairwise outranking situations (+0.75) is naturally given when we apply the COPELAND rule to the complete outranking digraph (see Listing 10.6 Line 6). When we apply the same rule to the sparse 4-tiled outranking digraph, we get a correlation of +0.72 (Line 12), and when applying the COPELAND rule to the optimised 4-tiled digraph, we still obtain a correlation of +0.71 (Line 18). These 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, for instance, we would get in a similar setting a complete outranking correlation of +0.86, a sparse 4-tiling correlation of +0.82, and an optimized sparse 4-tiling correlation of +0.81.

11.6 HPC quantiles ranking records

Following from the separability property of the q -tiles sorting of each action into each q -tiles class, the q -sorting algorithm may be safely split into as much threads as are multiple processing cores available in parallel. 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 outrankingdigraph.

Using the HPC platform of the University of Luxembourg (<https://hpc.uni.lu/>), the following run times for very big ranking problems could be achieved both:

- on Iris -skylake nodes with 28 cores Footnote[7], and
- on the 3TB -bigmem Gaia-183 node with 64 cores Footnote[8],

by running the cythonized python modules in an Intel compiled virtual Python 3.6.5 environment [GCC Intel(R) 17.0.1 –enable-optimizations c++ gcc 6.3 mode] on Debian 8 Linux.

Fig. 11.2 HPC-UL Ranking Performance Records (Spring 2018). On the big memory equipped Gaia-183 node, we were able to rank a million decision alternatives in about 2 minutes. We even could linearly rank up to 6 million decision alternatives in about 40 minutes.

\gtrless^q outranking relation order	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'

Part III

Evaluation and decision case studies

To be written

Chapter 12

Alice's best choice: A selection case study

Abstract To be written.
Footnote [19]

Fig. 12.1 Alice D., 19 years old German student finishing her secondary studies in Köln (Germany), desires to undertake foreign languages studies. She will probably receive her "Abitur" with satisfactory and/or good marks and wants to start her further studies thereafter. She would not mind staying in Köln, yet is ready to move elsewhere if necessary. The length of the higher studies do concern her, as she wants to earn her life as soon as possible. Her parents however agree to financially support her study fees as well as her living costs during her studies.



12.1 The decision problem

Alice has already identified 10 potential study programs.

In Table 12.1 we notice that Alice considers three *Graduate Interpreter* studies (8 or 9 Semesters), respectively in Köln, in Saarbrücken or in Heidelberg;

Table 12.1 The potential study programs

ID	Diploma	Institution	City
T-UD	Qualified translator (T)	University (UD)	Düsseldorf
T-FHK	Qualified translator (T)	Higher Technical School (FHK)	Köln
T-FHM	Qualified translator (T)	Higher Technical School (FHM)	München
I-FHK	Graduate interpreter (I)	Higher Technical School (FHK)	Köln
T-USB	Qualified translator (T)	University (USB)	Saarbrücken
I-USB	Graduate interpreter (I)	University (USB)	Saarbrücken
T-UHB	Qualified translator (T)	University (UHB)	Heidelberg
I-UHB	Graduate interpreter (I)	University (UHB)	Heidelberg
S-HKK	Specialized secretary (S)	Chamber of Commerce (HKK)	Köln
C-HKK	Foreign correspondent (C)	Chamber of Commerce (HKK)	Köln

and five *Qualified translator* studies (8 or 9 Semesters), respectively in Köln, in Düsseldorf, in Saarbrücken, in Heidelberg or in Munich. She also considers two short (4 Semesters) study programs at the Chamber of Commerce in Köln.

Four **decision objectives** of more or less equal importance are guiding Alice's choice:

1. *maximize* the attractiveness of the study place (GEO),
2. *maximize* the attractiveness of her further studies (LEA),
3. *minimize* her financial dependency on her parents (FIN),
4. *maximize* her professional perspectives (PRA).

The decision consequences Alice wishes to take into account for evaluating the potential study programs with respect to each of the four objectives are modelled by the following **coherent family of criteria** Footnote[26].

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
BC	Big City	Number of inhabitants (max)	GEO	3
AS	Studies	Attractiveness of the studies (max)	LEA	6
SF	Fees	Annual study fees (min)	FIN	2
LC	Living	Monthly living costs (min)	FIN	2
SL	Length	Length of the studies (min)	FIN	2
AP	Profession	Attractiveness of the profession (max)	PRA	2
AI	Income	Annual income after studying (max)	PRA	2
PR	Prestige	Occupational prestige (max)	PRA	2

Within each decision objective, the performance criteria are considered to be equisignificant. Hence, the four decision objectives show a same importance weight of 6 (see Table 12.2 Column 5).

12.2 The performance tableau

The actual evaluations of Alice's potential study programs are stored in a file named `AliceChoice.py` of `PerformanceTableau` format Footnote[21].

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
14     LC Monthly living costs      2
15     SL Study time               2
16     Total weight: 6.00 (3 criteria)
17     PRA: Professional aspect
18     AP Attractiveness of the profession          2
19     AI Annual professional income after studying 2
20     OP Occupational Prestige                  2
21     Total weight: 6.00 (3 criteria)
```

Details of the performance criteria may be consulted in a browser view (see Fig. 12.2 below).

```
1 >>> t.showHTMLCriteria()
```

It is worthwhile noticing in Fig. 12.2 that, on her subjective attractiveness scale of the study programs (criterion *AS*), Alice considers a performance differences of 7 points to be *considerable* and triggering, the case given, an outranking polarisation [BIS-2013]. Notice also the proportional *indifference* (1%) and *preference* (5%) discrimination thresholds shown on criterion *BC*-number of inhabitants.

Alice is subjectively evaluating the *Attractiveness* of the studies (criterion *AS*) on an ordinal scale from 0 (*weak*) to 10 (*excellent*). Similarly, she is subjectively evaluating the *Attractiveness* of the respective professions (criterion *AP*) on a three level ordinal scale from 0 (*weak*), 1 (*fair*) to 2 (*good*). Considering the *Occupational Prestige* (criterion *OP*), she looked up the SIOPS Footnote[20]. All the other evaluation data she found on the internet (see Fig. 12.3).

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.

In the following *heatmap view*, we may now consult Alice's performance evaluations.

```
1 >>> t.showHTMLPerformanceHeatmap(\n2 ...         colorLevels=5,Correlations=True,ndigits=0)
```

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

Notice by the way that evaluations on performance criteria to be *minimized*, 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.

Her ten potential study programs are ordered with the NETFLOWS ranking rule applied to the corresponding bipolar-valued outranking digraph Footnote[23]. *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 study programs for her appear to be studies at the Chamber of Commerce of Köln (*C – HKK, S – HKK*).

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* (*BC*, $\tau = -0.07$) as well as *Monthly living costs* (*LC*, $\tau = -0.04$) appear to be for her *not so significant* Footnote[27].

12.3 Building a best choice recommendation

Let us now have a look at the resulting pairwise outranking situations.

Listing 12.2 Computing Alice's outranking digraph

```

1 >>> from outrankingDigraphs import BipolarOutrankingDigraph
2 >>> dg = BipolarOutrankingDigraph(t)
3 >>> dg
4     *----- Object instance description -----*
5     Instance class      : BipolarOutrankingDigraph
6     Instance name       : rel_AliceChoice
7     Actions             : 10
8     Criteria            : 9
9     Size                : 67
10    Determinateness (%) : 73.91
11    Valuation domain   : [-1.00;1.00]
12 >>> dg.computeSymmetryDegree(Comments=True)
13     Symmetry degree of graph <rel_AliceChoice> : 0.49

```

From Alice's performance tableau we obtain 67 positively validated pairwise outranking situations in the digraph *dg*, supported by a 74% majority of criteria significance (see Listing 12.2 Line 9-10).

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 (see Fig. 12.4).

```

1 >>> dg.showHTMLRelationMap(\'
2 ...           tableTitle='Outranking relation map', \
3 ...           rankingRule='Copeland')

```

Outranking relation map

Ranking rule: Copeland

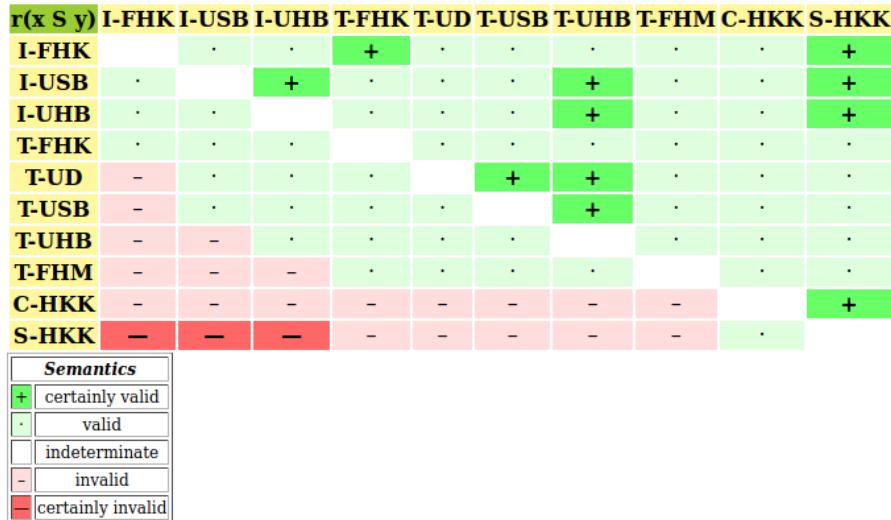


Fig. 12.4 COPELAND ranked outranking relation map

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, these polarisations appear in the last column and last row. We may inspect the occurrence of such polarisations as follows.

Listing 12.3 Polarised outranking situations

```

1 >>> dg.showVetos()
2      ----- Veto situations ---
3      number of veto situations : 3
4      1: r(S-HKK >= I-FHK) = -0.17
5          criterion: AS
6          Considerable performance difference : -7.00
7          Veto discrimination threshold       : -7.00
8          Polarisation: r(S-HKK >= I-FHK) = -0.17 ==> -1.00
9      2: r(S-HKK >= I-USB) = -0.17
10         criterion: AS
11         Considerable performance difference : -7.00
12         Veto discrimination threshold       : -7.00
13         Polarisation: r(S-HKK >= I-USB) = -0.17 ==> -1.00
14      3: r(S-HKK >= I-UHB) = -0.17
15         criterion: AS
16         Considerable performance difference : -7.00

```

```

17      Veto discrimination threshold      : -7.00
18      Polarisation: r(S-HKK >= I-UHB) = -0.17 ==> -1.00
19      *---- Counter-veto situations ---
20      number of counter-veto situations : 3
21      1: r(I-FHK >= S-HKK) = 0.83
22          criterion: AS
23          Considerable performance difference : 7.00
24          Counter-veto threshold           : 7.00
25          Polarisation: r(I-FHK >= S-HKK) = 0.83 ==> +1.00
26      2: r(I-USB >= S-HKK) = 0.17
27          criterion: AS
28          Considerable performance difference : 7.00
29          Counter-veto threshold           : 7.00
30          Polarisation: r(I-USB >= S-HKK) = 0.17 ==> +1.00
31      3: r(I-UHB >= S-HKK) = 0.17
32          criterion: AS
33          Considerable performance difference : 7.00
34          Counter-veto threshold           : 7.00
35          Polarisation: r(I-UHB >= S-HKK) = 0.17 ==> +1.00

```

In Listing 12.3, we see that *considerable* performance differences concerning the *Attractiveness of the studies* (AS criterion) are indeed observed between the *Specialised Secretary* study programm offered in Köln and the *Graduate Interpreter* study programs offered in Köln, Saarbrücken and Heidelberg. They polarise, hence, three *more or less invalid* outranking situations to *certainly invalid* (Lines 8, 13, 18) and corresponding three *more or less valid* converse outranking situations to *certainly valid* ones (Lines 25, 30, 35).

We may finally notice in the relation map, shown in Fig. 12.4, that the four best-ranked study programs, *I – FHK*, *I – USB*, *I – UHB* and *T – FHK*, are in fact CONDORCET winners (see Listing 12.4 Line 2), i.e. they are all four *indifferent* one of the other **and** positively *outrank* all other alternatives, a result confirmed below by our best choice recommendation (Line 8).

Listing 12.4 Alice's best choice recommendation

```

1 >>> dg.computeCondorcetWinners()
2 ['I-FHK', 'I-UHB', 'I-USB', 'T-FHK']
3 >>> dg.showBestChoiceRecommendation()
4 Best choice recommendation(s) (BCR)
5   (in decreasing order of determinateness)
6   Credibility domain: [-1.00,1.00]
7 === >> potential best choice(s)
8   choice           : ['I-FHK', 'I-UHB', 'I-USB', 'T-FHK']
9   independence     : 0.17
10  dominance        : 0.08
11  absorbency       : -0.83
12  covering (%)    : 62.50
13  determinateness (%) : 68.75
14  most credible action(s) = {'I-FHK': 0.75, 'T-FHK': 0.17,
15                                'I-USB': 0.17, 'I-UHB': 0.17}
16 === >> potential worst choice(s)
17   choice           : ['C-HKK', 'S-HKK']
18   independence     : 0.50

```

```

19      dominance          : -0.83
20      absorbency         : 0.17
21      covered (%)        : 100.00
22      determinateness (%) : 58.33
23      most credible action(s) = {'S-HKK': 0.17, 'C-HKK': 0.17}

```

Most credible best choice among the four best-ranked study programs eventually becomes the *Graduate Interpreter* study program at the Technical High School in Köln (see Listing 12.4 Line 14) supported by a $(0.75 + 1)/2.0 = 87.5\%(18/24)$ majority of global criteria significance Footnote[24].

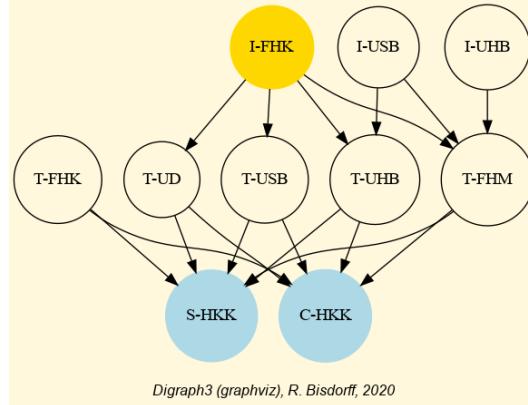
In the relation map, shown in Fig. 12.4, we see in the left lower corner that the *asymmetric part* of the outranking relation, i.e. the corresponding *strict* outranking relation, is actually *transitive* (see Line 2 below). Hence, a *graphviz* drawing of its *skeleton*, oriented by the previous *best*, respectively *worst* choice, may well illustrate our *best choice recommendation*.

```

1 >>> dgcd = ~(-dg)
2 >>> dgcd.isTransitive()
3     True
4 >>> dgcd.closeTransitive(Reverse=True, InSite=True)
5 >>> dgcd.exportGraphViz('aliceBestChoice', \
6 ...                         bestChoice=['I-FHK'],
7 ...                         worstChoice=['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

```

Fig. 12.5 In 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).



Let us now, for instance, check the pairwise outranking situations observed between the first and second-ranked alternative, i.e. *Graduate Interpreter* studies in Köln versus *Graduate Interpreter* studies in Saarbrücken (see *I-FHK* and *I-USB* in Fig. 12.3).

```

1 >>> dg.showHTMLPairwiseOutrankings('I-FHK', 'I-USB')

```

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

Fig. 12.6 Comparing the first and second best-ranked study programs. The Köln alternative is performing *at least as well 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).

In a similar way, we may finally compute a *weak ranking* of all the potential study programs with the help of the `RankingByChoosingDigraph` constructor, who computes a bipolar ranking by conjointly *best-choosing* and *last-rejecting* [BIS-1999].

Listing 12.5 Weakly ranking by bipolar best-choosing and last-rejecting

```

1 >>> from transitiveDigraphs import \
2 ...           RankingByChoosingDigraph
3
4 >>> rbc = RankingByChoosingDigraph(dg)
5 >>> rbc.showRankingByChoosing()
6   Ranking by Choosing and Rejecting
7   1st ranked ['I-FHK']
8   2nd ranked ['I-USB']
9   3rd ranked ['I-UHB']
10  4th ranked ['T-FHK']
11  5th ranked ['T-UD']
12  5th last ranked ['T-UD']
13  4th last ranked ['T-UHB', 'T-USB']
14  3rd last ranked ['T-FHM']
15  2nd last ranked ['C-HKK']
16  1st last ranked ['S-HKK']
```

In Listing 12.5, 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 *Translater* 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?

12.4 Robustness analysis

Alice considers her four decision objectives as being *more or less* equally important. Here we have, however, allocated *strictly equal* importance weights with *strictly equi-significant* criteria per objective. How robust is our previous best choice recommendation when, now, we would consider the importance of the objectives and, hence, the significance of the respective performance criteria to be *more or less uncertain*?

To answer this question, we will consider the respective criteria significance weights w_j for $j = 1, \dots, 9$ to be *triangular random variables* in the range 0 to $2w_j$ with mode = w_j . We may compute a corresponding 90%-*confident* outranking digraph with the help of the `ConfidentBipolarOutrankingDigraph` constructor Footnote[22].

Listing 12.6 Computing the 90% confident outranking digraph

```

1 >>> from outrankingDigraphs import \
```

```

2     ...
3     ConfidentBipolarOutrankingDigraph
4 >>> cdg = ConfidentBipolarOutrankingDigraph(t,\n5 ...     distribution='triangular',confidence=90.0)
6
7 >>> cdg
8 *----- Object instance description -----*
9 Instance class      : ConfidentBipolarOutrankingDigraph
10 Instance name       : rel_AliceChoice_CLT
11 Actions             : 10
12 Criteria            : 9
13 Size                : 44
14 Valuation domain    : [-1.00;1.00]
15 Uncertainty model   : triangular(a=0,b=2w)
16 Likelihood domain   : [-1.0;+1.0]
17 Confidence level    : 90.0%
18 Confident majority   : 14/24 (58.3%)
19 Determinateness (%) : 68.19

```

Of the original 67 valid outranking situations, we retain 44 outranking situations as being 90%-confident (see Listing ?? Line 13). The corresponding 90%-*confident* **qualified majority** of criteria significance amounts to $14/24 = 58.3\%$ (Line 18).

Concerning now a 90%-confident best choice recommendation, we are lucky.

Listing 12.7 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',
10                  'T-FHK','T-FHM']
11 independence    : 0.00
12 dominance       : 0.42
13 absorbency      : 0.00
14 covering (%)    : 20.00
15 determinateness (%) : 61.25
16 - most credible action(s) = { 'I-FHK': 0.75, }

```

The *Graduate Interpreter* studies in Köln remain indeed a 90%-confident CONDORCET winner (see Fig. 12.7 Line 2). Hence, the same study program also remains our 90%-confident most credible best choice supported by a continual $18/24(87.5\%)$ majority of the global criteria significance (see Lines 9-10 and 16).

When previously comparing the two best-ranked study programs (see Fig. 12.6), we have observed that *I – FHK* actually positively outranks *I – USB* on all four decision objectives. When admitting equi-significant criteria significances per objective, this outranking situation is hence valid independently of the importance weights Alice may allocate to each of her decision objectives.

We may compute these **unopposed** outranking situations Footnote[25] with help of the `UnOpposedBipolarOutrankingDigraph` constructor (see Chapter ??).

Listing 12.8 Computing the unopposed outranking situations

```

1 >>> from outrankingDigraphs import
2     UnOpposedBipolarOutrankingDigraph
3 >>> uop = UnOpposedBipolarOutrankingDigraph(t)
4 >>> uop
5     *----- Object instance description -----*
6     Instance class      : UnOpposedBipolarOutrankingDigraph
7     Instance name       : AliceChoice_unopposed_outrankings
8     Actions             : 10
9     Criteria            : 9
10    Size                : 28
11    Oppositeness (%)   : 58.21
12    Determinateness (%) : 62.94
13    Valuation domain   : [-1.00;1.00]
14 >>> uop.isTransitive()
14     True

```

We keep 28 out the 67 standard outranking situations, which leads to an *oppositeness degree* of $(1.0 - 28/67) = 58.21\%$ (see Listing 12.8 Line 10). Remarkable furthermore is that this unopposed outranking digraph *uop* is actually *transitive*, i.e. modelling a *partial ranking* of the study programs (Line 14).

We may hence make use of the `exportGraphViz()` method of the `TransitiveDigraph` class for drawing the corresponding partial ranking.

```

1 >>> from transitiveDigraphs import TransitiveDigraph
2 >>> TransitiveDigraph.exportGraphViz(uop,\ 
3 ...           fileName='choice_unopposed')
4     *---- exporting a dot file for GraphViz tools -----*
5     Exporting to choice_unopposed.dot
6     dot -Grankdir=TB -Tpng choice_unopposed.dot -o
         choice_unopposed.png

```

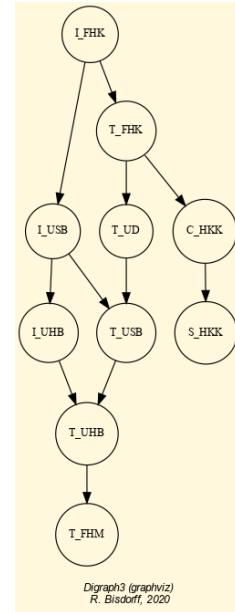


Fig. 12.7 Unopposed partial ranking of the potential study programs. Again, when *equi-significant* performance criteria are assumed per decision objective, we observe in here that *I – FHK* remains the stable best choice, *independently* of the actual importance weights that Alice may wish to allocate to her four decision objectives.

In view of her performance tableau in Fig. 12.3, *Graduate Interpreter* studies at the Technical High School Köln, thus, represent definitely **Alice's very best choice**.

For further reading about the *Rubis Best Choice* methodology, one may consult in [BIS-2015] the study of a *real decision aid case* about choosing a best poster in a scientific conference.

Chapter 13

The best academic Computer Science Depts: A ranking case study

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 Footnote[36]. 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 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 ordinal significance. A third Section is finally devoted to introduce quality measures for qualifying ranking results.

13.1 The THE performance tableau

For our tutorial purpose, an extract of the published THE University rankings 2016 by computer science subject data, concerning the 75 first-ranked academic Institutions, is stored in a file named `the_cs_2016.py` of PerformanceTableau format Footnote[37].

Listing 13.1 The 2016 THE World University Ranking by CS subject

```
1 >>> from perfTabs import PerformanceTableau
2 >>> t = PerformanceTableau('the_cs_2016')
3 >>> t
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 actions, in our case here, are the 75 THE best-ranked CS Departments, 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, we will illustrate below how to write small Python scripts for printing out its content.

Listing 13.2 Printing the potential decision actions

```

1 >>> for x in t.actions:
2 ...     print('%s:\t%s (%s)' %\
3 ...           (x,t.actions[x]['name'],t.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)
24 frei: University of Freiburg (DE)
25 git: Georgia Institute of Technology (US)
26 glas: University of Glasgow (UK)
27 hels: University of Helsinki (FI)
28 hkpu: Hong Kong Polytechnic University (CN)
29 hkst: Hong Kong University of Science and Technology (HK)
30 hku: Hong Kong University (HK)
31 humb: Berlin Humboldt University (DE)
32 icl: Imperial College London (UK)
33 indiss: Indian Institute of Science (IN)
34 itmo: ITMO University (RU)
35 kcl: King's College London (UK)
36 kist: Korea Adv. Institute of Science and Technology (KR)
37 kit: Karlsruhe Institute of Technology (DE)

```

```

38 kth: KTH Royal Institute of Technology (SE)
39 kuj: Kyoto University (JP)
40 kul: Catholic University Leuven (BE)
41 lms: Lomonosov Moscow State University (RU)
42 man: University of Manchester (UK)
43 mcp: University of Maryland College Park (US)
44 mel: University of Melbourne (AU)
45 mil: Polytechnic University of Milan (IT)
46 mit: Massachusetts Institute of Technology (US)
47 naji: Nanjing University (CN)
48 ntu: Nanyang Technological University of Singapore (SG)
49 ntw: National Taiwan University (TW)
50 nyu: New York University (US)
51 oxf: University of Oxford (UK)
52 pud: Purdue University (US)
53 qut: Queensland University of Technology (AU)
54 rcu: Rice University (US)
55 rwth: RWTH Aachen University (DE)
56 shji: Shanghai Jiao Tong University (CN)
57 sing: National University of Singapore (SG)
58 sou: University of Southampton (UK)
59 stut: University of Stuttgart (DE)
60 tech: Technion - Israel Institute of Technology (IL)
61 tlavu: Tel Aviv University (IR)
62 tsu: Tsinghua University (CN)
63 tub: Technical University of Berlin (DE)
64 tud: Technical University of Darmstadt (DE)
65 tum: Technical University of Muenchen (DE)
66 ucl: University College London (UK)
67 ued: University of Edinburgh (UK)
68 uiu: University of Illinois at Urbana-Champaign (US)
69 unlu: University of Luxembourg (LU)
70 unsw: University of New South Wales (AU)
71 unt: University of Toronto (CA)
72 uta: University of Texas at Austin (US)
73 utj: University of Tokyo (JP)
74 utw: University of Twente (NL)
75 uwa: University of Waterloo (CA)
76 wash: University of Washington (US)
77 wtu: Vienna University of Technology (AUS)
78 zhej: Zhejiang University (CN)

```

The THE authors base their ranking decisions on five objectives.

Listing 13.3 The THE ranking objectives

```

1 >>> for obj in t.objectives:
2 ...     print('%s: %s (%.1f%%),\n\t%s' %
3 ...           (obj,t.objectives[obj]['name'],
4 ...            t.objectives[obj]['weight'],
5 ...            t.objectives[obj]['comment']))\
6 ...
7
8 Teaching: Best learning environment (30.0%),
9 Reputation survey; Staff-to-student ration;

```

```

10 Doctorate-to-student ratio,
11 Doctorate-to-academic-staff ratio,
12 Institutional income.
13 Research: Highest volume and repustation (30.0%),
14 Reputation survey;
15 Research income;
16 Research productivity
17 Citations: Highest research influence (27.5%),
18 Impact.
19 International outlook: Most international staff,
20 students and research (7.5%),
21 Proportions of international students;
22 Proportions of international staff;
23 International collaborations.
24 Industry income: Best knowledge transfer (5.0%),
25 Volume.

```

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 does, unfortunately, not publish the detail of their performance assessments for grading CS Depts with respect to each one of the five ranking objectives Footnote[39]. 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 t.criteria:
2 ...     print('%s:\t%s, %s (%.1f%%)' %\
3 ...         (g,t.criteria[g]['name'],t.criteria[g]['comment'],\
4 ...          t.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 gind: 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 grade obtained by each University with the corresponding *quantile* observed in the cumulative distribution of the performance grades obtained by all the surveyed institutions Footnote[40]. The THE ranking is eventually determined by an *overall score* per University which corresponds to the weighted average of these five criteria quantiles (see Listing 13.4 below).

Listing 13.4 Computing the THE overall scores

```

1 >>> theScores = []

```

```

2 >>> for x in t.actions:
3 ...     xscore = Decimal('0')
4 ...     for g in t.criteria:
5 ...         xscore += t.evaluation[g][x] * \
6 ...             (t.criteria[g]['weight']/Decimal('100'))
7 ...     theScores.append((xscore,x))

```

In Listing 13.5 (Lines 15-16) below, we may thus notice that, in the 2016 edition of the THE World University rankings by CS subject, the Swiss Federal Institute of Technology Zürich is first-ranked with an overall score of 92.9; followed by the California Institute of Technology (overall score: 92.4) Footnote[38].

Listing 13.5 Printing the ranked performance table

```

1 >>> theScores.sort(reverse = True)
2 >>> print('## Univ \tgch 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 t.criteria:
10 ...         print('.1f' % (t.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
37    23: uwa       75.3    82.6   91.3   72.9   41.5    80.0
38    24: nyu       71.1    77.4   99.4   78.0   39.8    79.7
39    25: uta       72.6    85.3   99.6   31.6   49.7    79.6
40    26: kit       73.8    85.5   84.4   41.3   76.8    77.9
41    27: bju      83.0    85.3   70.1   30.7   99.4    77.0

```

42	28: csb	65.6	70.9	94.8	72.9	74.9	76.2
43	29: rwth	77.8	85.0	70.8	43.7	89.4	76.1
44	30: hku	77.0	73.0	77.0	96.8	39.5	75.4
45	31: pud	76.9	84.8	70.8	58.1	56.7	75.2
46	32: kist	79.4	88.2	64.2	31.6	92.8	74.9
47	33: kcl	45.5	94.6	86.3	95.1	38.3	74.8
48	34: chku	64.1	69.3	94.7	75.6	49.9	74.2
49	35: epfr	81.7	60.6	78.1	85.3	62.9	73.7
50	36: dut	64.1	78.3	76.3	69.8	90.1	73.4
51	37: tub	66.2	82.4	71.0	55.4	99.9	73.3
52	38: utj	92.0	91.7	48.7	25.8	49.6	72.9
53	39: cir	68.8	64.6	93.0	65.1	40.4	72.5
54	40: ntw	81.5	79.8	66.6	25.5	67.6	72.0
55	41: anu	47.2	73.0	92.2	90.0	48.1	70.6
56	42: rcu	64.1	53.8	99.4	63.7	46.1	69.8
57	43: mel	56.1	70.2	83.7	83.3	50.4	69.7
58	44: lms	81.5	68.1	61.0	31.1	87.8	68.4
59	45: ens	71.8	40.9	98.7	69.6	43.5	68.3
60	46: wtu	61.8	73.5	73.7	51.9	62.2	67.9
61	47: tech	54.9	71.0	85.1	51.7	40.1	67.1
62	48: bro	58.5	54.9	96.8	52.3	38.6	66.5
63	49: man	63.5	71.9	62.9	84.1	42.1	66.3
64	50: zhej	73.5	70.4	60.7	22.6	75.7	65.3
65	51: frei	54.2	51.6	89.5	49.7	99.9	65.1
66	52: unsw	60.2	58.2	70.5	87.0	44.3	63.6
67	53: kuj	75.4	72.8	49.5	28.3	51.4	62.8
68	54: sou	48.2	60.7	75.5	87.4	43.2	62.1
69	55: shJi	66.9	68.3	62.4	22.8	38.5	61.4
70	56: itmo	58.0	32.0	98.7	39.2	68.7	60.5
71	57: kul	35.2	55.8	92.0	46.0	88.3	60.5
72	58: glas	35.2	52.5	91.2	85.8	39.2	59.8
73	59: utw	38.2	52.8	87.0	69.0	60.0	59.4
74	60: stut	54.2	60.6	61.1	36.3	97.8	58.9
75	61: naji	51.4	76.9	48.8	39.7	74.4	58.6
76	62: tud	46.6	53.6	75.9	53.7	66.5	58.3
77	63: unlu	35.2	44.2	87.4	99.7	54.1	58.0
78	64: qut	45.5	42.6	82.8	75.2	63.0	58.0
79	65: hkpu	46.8	36.5	91.4	73.2	41.5	57.7
80	66: albt	39.2	53.3	69.9	91.9	75.4	57.6
81	67: mil	46.4	64.3	69.2	44.1	38.5	57.5
82	68: hels	48.8	49.6	80.4	50.6	39.5	57.4
83	69: cihk	42.4	44.9	80.1	76.2	67.9	57.3
84	70: tlavu	34.1	57.2	89.0	45.3	38.6	57.2
85	71: indis	56.9	76.1	49.3	20.1	41.5	57.0
86	72: ariz	28.4	61.8	84.3	59.3	42.0	56.8
87	73: kth	44.8	42.0	83.6	71.6	39.2	56.4
88	74: humb	48.4	31.3	94.7	41.5	45.5	55.3
89	75: eind	32.4	48.4	81.5	72.2	45.8	54.4

It is important to notice that a ranking by weighted average scores requires *com-mensurable ranking criteria* of precise decimal significance and on which a precise decimal performance grading is given. It is very unlikely that the THE 2016 performance assessments indeed verify these conditions. Here we show how to relax

these methodological requirements –precise commensurable criteria and numerical assessments– by following instead an epistemic bipolar-valued logic based ranking methodology.

13.2 Ranking with multiple criteria of ordinal significance

Let us, first, have a critical look at the THE performance criteria.

```
>>> t.showHTMLCriteria(Sorted=False)
```

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

Considering a very likely imprecision of the performance grading procedure, followed by some potential violation of uniform distributed quantile classes, we assume here that a performance quantile difference of up to $\text{abs}(2.5)\%$ is *insignificant*, whereas a difference of 5% warrants a *clearly better*, resp. *clearly less good* performance. With quantiles 94%, resp. 87.3%, Oxford's CS teaching environment, for instance, is thus clearly better evaluated than that of the MIT (see Listing 13.5 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 a polarisation of a pairwise outranking, respectively a pairwise outranked situation [BIS-2013].

The effect of these performance discrimination thresholds on the preference modelling may be inspected as follows.

Listing 13.6 Inspecting the performance discrimination thresholds

```
>>> t.showCriteria()
----- criteria -----
gtch 'Teaching'
    Scale = (Decimal('0.00'), Decimal('100.00'))
    Weight = 0.300
    Threshold ind : 2.50 + 0.00x ; percentile: 8.07
    Threshold pref : 5.00 + 0.00x ; percentile: 15.75
```

```

8      Threshold veto : 60.00 + 0.00x ; percentile: 99.75
9      gres 'Research'
10     Scale = (Decimal('0.00'), Decimal('100.00'))
11     Weight = 0.300
12     Threshold ind : 2.50 + 0.00x ; percentile: 7.86
13     Threshold pref : 5.00 + 0.00x ; percentile: 16.14
14     Threshold veto : 60.00 + 0.00x ; percentile: 99.21
15     gcit 'Citations'
16     Scale = (Decimal('0.00'), Decimal('100.00'))
17     Weight = 0.275
18     Threshold ind : 2.50 + 0.00x ; percentile: 11.82
19     Threshold pref : 5.00 + 0.00x ; percentile: 22.99
20     Threshold veto : 60.00 + 0.00x ; percentile: 100.00
21     gint 'International outlook'
22     Scale = (Decimal('0.00'), Decimal('100.00'))
23     Weight = 0.075
24     Threshold ind : 2.50 + 0.00x ; percentile: 6.45
25     Threshold pref : 5.00 + 0.00x ; percentile: 11.75
26     gind 'Industry income'
27     Scale = (Decimal('0.00'), Decimal('100.00'))
28     Weight = 0.050
29     Threshold ind : 2.50 + 0.00x ; percentile: 11.82
30     Threshold pref : 5.00 + 0.00x ; percentile: 21.51

```

Between 6% and 12% of the observed quantile differences are, thus, considered to be *insignificant*. Similarly, between 77% and 88% are considered to be *significant*. Less than 1% correspond to *considerable* quantile differences on both the *Teaching* and *Research* criteria; actually triggering an epistemic polarisation effect [BIS-2013].

Beside the likely imprecise performance discrimination, the *precise decimal significance weights*, as allocated by the THE authors to the five ranking criteria are, as well, quite *questionable*. Criteria significance weights may carry usually hidden strategies for rendering the performance evaluations commensurable in view of a numerical computation of the overall ranking scores. The eventual ranking result is thus as much depending on the precise values of the given criteria significance weights as, vice versa, the given precise significance weights are depending on the subjectively expected and accepted ranking results Footnote[42]. We will therefore drop such precise weights and, instead, only require a corresponding criteria significance preorder: 'gtch' = 'gres' \triangleleft 'gcit' \triangleleft 'gint' \triangleleft 'gind'. *Teaching environment* and *Research volume and reputation* are equally considered most important, followed by *Research influence*. Then comes *International outlook in staff, students and research* and, least important finally, *Industry income and innovation*.

Both these working hypotheses: performance discrimination thresholds and solely ordinal criteria significance, give us way to a ranking methodology based on *robust pairwise outranking* situations (see Chapter 19 and [BIS-2004b]):

- We say that CS Dept x *robustly outranks* CS Dept y when x positively outranks y with **all** significance weight vectors that are compatible with the significance preorder: 'gtch' = 'gres' \triangleleft 'gcit' \triangleleft 'gint' \triangleleft 'gind';

- 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' \wedge 'gcit' \wedge 'gint' \wedge 'gind';
- Otherwise, CS Depts x and y are considered to be *incomparable*.

A corresponding digraph constructor is provided by the `RobustOutrankingDigraph` class.

Listing 13.7 Computing the robust outranking digraph

```

1 >>> from outrankingDigraphs import RobustOutrankingDigraph
2
3 >>> rdg = RobustOutrankingDigraph(t)
4
5 *----- Object instance description -----*
6   Instance class      : RobustOutrankingDigraph
7   Instance name       : robust_the_cs_2016
8   Actions             : 75
9   Criteria            : 5
10  Size                : 2993
11  Determinateness (%) : 78.16
12  Valuation domain    : [-1.00;1.00]
13  >>> rdg.computeIncomparabilityDegree(Comments=True)
14  Incomparability degree (%) of digraph <robust_the_cs_2016>:
15  links x<->y: 2775, incomparable: 102, comparable: 2673
16  (incomparable/links) = 0.037
17  >>> rdg.computeTransitivityDegree(Comments=True)
18  Transitivity degree of digraph <robust_the_cs_2016>:
19  triples x>y>z: 405150, closed: 218489, open: 186661
20  (closed/triples) = 0.539
21  >>> rdg.computeSymmetryDegree(Comments=True)
22  Symmetry degree (%) of digraph <robust_the_cs_2016>:
23  arcs x>y: 2673, symmetric: 320, asymmetric: 2353
24  (symmetric/arcs) = 0.12

```

In the resulting digraph instance rdg (see Listing 13.7 Line 8), we observe 2993 such robust pairwise outranking situations validated with a mean significance of 78% (Line 9). Unfortunately, in our case here, they do not deliver any complete linear ranking relation. The robust outranking digraph rdg contains in fact 102 incomparability situations (3.7%, Line 13); nearly half of its transitive closure is missing (46.1%, Line 18) and 12% of the positive outranking situations correspond in fact to symmetric indifference situations (Line 22).

Worse even, the digraph rdg admits furthermore a high number of outranking circuits.

Listing 13.8 Inspecting outranking circuits

```

1 >>> rdg.computeChordlessCircuits()
2 >>> rdg.showChordlessCircuits()
3
4 *---- Chordless circuits ----*
5   145 circuits.
6   1: ['albt', 'unlu', 'ariz', 'hels'] , credibility : 0.300
7   2: ['albt', 'tlavu', 'hels'] , credibility : 0.150
8   3: ['anu', 'man', 'itmo'] , credibility : 0.250

```

```

8      4:  ['anu', 'zhej', 'rcu'] , credibility : 0.250
9      ...
10     ...
11    82:  ['csb', 'epfr', 'rwth'] , credibility : 0.250
12    83:  ['csb', 'epfr', 'pud', 'nyu'] , credibility : 0.250
13    84:  ['csd', 'kcl', 'kist'] , credibility : 0.250
14      ...
15      ...
16    142:  ['kul', 'qut', 'mil'] , credibility : 0.250
17    143:  ['lms', 'rcu', 'tech'] , credibility : 0.300
18    144:  ['mil', 'stut', 'qut'] , credibility : 0.300
19    145:  ['mil', 'stut', 'tud'] , credibility : 0.300

```

Among the 145 detected robust outranking circuits reported in Listing 13.8, we notice, for instance, two outranking circuits of length 4 (see circuits 1 and 83). Let us explore below the bipolar-valued robust outranking characteristics $r(x \succsim y)$ of the first circuit.

Listing 13.9 Showing the relation table with stability denotation

```

1 >>> rdg.showRelationTable(actionsSubset=
2     ['albt','unlu','ariz','hels'],\
3     Sorted=False)
4
5 * ----- Relation Table -----
6 r/(stab) | 'albt' 'unlu' 'ariz' 'hels'
7   -----|-----
8   'albt' | +1.00 +0.30 +0.00 +0.00
9     | (+4)  (+2) (-1) (-1)
10  'unlu' | +0.00 +1.00 +0.40 +0.00
11  | (+0)  (+4) (+2) (-1)
12  'ariz' | +0.00 -0.12 +1.00 +0.40
13  | (+1)  (-2) (+4) (+2)
14  'hels' | +0.45 +0.00 -0.03 +1.00
15  | (+2)  (+1) (-2) (+4)
16 Valuation domain: [-1.0; 1.0]
17 Stability denotation semantics:
18 +4|-4 : unanimous outranking | outranked situation;
19 +2|-2 : outranking | outranked situation validated
20           with all potential significance weights that are
21           compatible with the given significance preorder;
22 +1|-1 : validated outranking | outranked situation with
23           the given significance weights;
           0   : indeterminate relational situation.

```

In Listing 13.10, we may 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: $'gtch' = 'gres' \wedge 'gcit' \wedge 'gint' \wedge 'gind'$. Notice also the $(+1--1)$ 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 weight vector, but not with all potential weight vectors compatible with the given significance preorder. All these outranking situations are hence put into *doubt* ($r(x \succsim y) = 0.00$) and the corre-

sponding 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 the Chapter on ranking with multiple criteria (Chapter 8) we have seen now several potential heuristic ranking rules that may be applied to rank from a pairwise outranking digraph; yet, delivering all 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 Footnote[41]. Its complexity in $O(n^2)$ is indeed quite tractable and, by avoiding potential *tyranny of short majority* effects, the NETFLOWS rule specifically takes the ranking criteria significance into a more fairly balanced account.

The NETFLOWS ranking result of the CS Depts may be computed explicitly as follows.

Listing 13.10 Showing the relation table with stability denotation

```

1 >>> nfRanking = rdg.computeNetFlowsRanking()
2 >>> nfRanking
3 ['ethz', 'calt', 'mit', 'oxf', 'cmel', 'git', 'epfl',
4  'icl', 'cou', 'tum', 'wash', 'sing', 'hkst', 'ucl',
5  'uiu', 'unt', 'ued', 'ntu', 'mcp', 'csd', 'cbu',
6  'uta', 'tsu', 'nyu', 'uwa', 'csb', 'kit', 'utj',
7  'bju', 'kcl', 'chku', 'kist', 'rwth', 'pud', 'epfr',
8  'hku', 'rcu', 'cir', 'dut', 'ens', 'ntw', 'anu',
9  'tub', 'mel', 'lms', 'bro', 'frei', 'wtu', 'tech',
10 'itmo', 'zhej', 'man', 'kuj', 'kul', 'unsw', 'glas',
11 'utw', 'unlu', 'naji', 'sou', 'hkpu', 'qut', 'humb',
12 'shji', 'stut', 'tud', 'tlavu', 'cihk', 'albt', 'indis',
13 'ariz', 'kth', 'hels', 'eind', 'mil']

```

We actually obtain a very similar ranking result as the one obtained with the THE overall 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.

We may print out the difference between the overall scores based THE ranking and our NETFLOWS ranking with the following short Python script, where we make use of an ordered Python dictionary with net flow scores, stored in the *rdg.netFlowsRankingDict* attribute by the previous computation.

Listing 13.11 Comparing the robust NETFLOWS ranking with the THE ranking

```

1 >>> # rdg.netFlowsRankingDict: ordered dictionary with net flow
2 >>> # scores stored in rdg by the computeNetFlowsRanking() method
3 >>> # theScores = [(xScore_1,x_1), (xScore_2,x_2),...]
4 >>> # is sorted in decreasing order of xscores_i
5 >>> print(\n
6 ... ' NetFlows ranking    gtch    gres    gcit    gint    gind    THE
7 ... ' ranking')

```

```

8 >>> for i in range(75):
9 ...     x = nfRanking[i]
10 ...    xScore = rdg.netFlowsRankingDict[x]['netFlow']
11 ...    thexScore,thex = theScores[i]
12 ...    print('%2d: %s (%.2f) ' % (i+1,x,xScore), end=' \t')
13 ...    for g in rdg.criteria:
14 ...        print('%.1f ' % (t.evaluation[g][x]),end=' ')
15 ...    print(' %s (%.2f)' % (thex,thexScore) )
16
17     NetFlows ranking      gtch      gres      gcit      gint      gind      THE ranking
18     1: ethz (116.95)    89.2     97.3     97.1     93.6     64.1     ethz (92.88)
19     2: calt (116.15)    91.5     96.0     99.8     59.1     85.9     calt (92.42)
20     3: mit (112.72)     87.3     95.4     99.4     73.9     87.5     oxf (92.20)
21     4: oxf (112.00)     94.0     92.0     98.8     93.6     44.3     mit (92.06)
22     5: cmel (101.60)    88.1     92.3     99.4     58.9     71.1     git (89.88)
23     6: git (93.40)      87.2     99.7     91.3     63.0     79.5     cmel (89.43)
24     7: epfl (90.88)     86.3     91.6     94.8     97.2     42.7     icl (89.00)
25     8: icl (90.62)      90.1     87.5     95.1     94.3     49.9     epfl (88.86)
26     9: cou (84.60)      81.6     94.1     99.7     55.7     45.7     tum (87.70)
27    10: tum (80.42)      87.6     95.1     87.9     52.9     95.1     sing (86.86)
28    11: wash (76.28)     84.4     88.7     99.3     57.4     41.2     cou (86.59)
29    12: sing (73.05)     89.9     91.3     83.0     95.3     50.6     ucl (86.05)
30    13: hkst (71.05)     74.3     92.0     96.2     84.4     55.8     wash (85.60)
31    14: ucl (66.78)      85.5     90.3     87.6     94.7     42.4     hkst (85.47)
32    15: uiu (64.80)      85.0     83.1     99.2     51.4     42.2     ntu (85.46)
33    16: unt (62.65)      79.9     84.4     99.6     77.6     38.4     ued (85.03)
34    17: ued (58.67)      85.7     85.3     89.7     95.0     38.8     unt (84.42)
35    18: ntu (57.88)      76.6     87.7     90.4     92.9     86.9     uiu (83.67)
36    19: mcp (54.08)      79.7     89.3     94.6     29.8     51.7     mcp (81.53)
37    20: csd (46.62)      75.2     81.6     99.8     39.7     59.8     cbu (81.25)
38    21: cbu (44.27)      81.2     78.5     94.7     66.9     45.7     tsu (80.91)
39    22: uta (43.27)      72.6     85.3     99.6     31.6     49.7     csd (80.45)
40    23: tsu (42.42)      88.1     90.2     76.7     27.1     85.9     uwa (80.02)
41    24: nyu (35.30)      71.1     77.4     99.4     78.0     39.8     nyu (79.72)
42    25: uwa (28.88)      75.3     82.6     91.3     72.9     41.5     uta (79.61)
43    26: csb (18.18)      65.6     70.9     94.8     72.9     74.9     kit (77.94)
44    27: kit (16.32)      73.8     85.5     84.4     41.3     76.8     bju (77.04)
45    28: utj (15.95)      92.0     91.7     48.7     25.8     49.6     csb (76.23)
46    29: bju (15.45)      83.0     85.3     70.1     30.7     99.4     rwth (76.06)
47    30: kcl (11.95)      45.5     94.6     86.3     95.1     38.3     hku (75.41)
48    31: chku (9.43)      64.1     69.3     94.7     75.6     49.9     pud (75.17)
49    32: kist (7.30)      79.4     88.2     64.2     31.6     92.8     kist (74.94)
50    33: rwth (5.00)      77.8     85.0     70.8     43.7     89.4     kcl (74.81)
51    34: pud (2.40)       76.9     84.8     70.8     58.1     56.7     chku (74.23)
52    35: epfr (-1.70)     81.7     60.6     78.1     85.3     62.9     epfr (73.71)
53    36: hku (-3.83)      77.0     73.0     77.0     96.8     39.5     dut (73.44)
54    37: rcu (-6.38)      64.1     53.8     99.4     63.7     46.1     tub (73.25)
55    38: cir (-8.20)      68.8     64.6     93.0     65.1     40.4     utj (72.92)
56    39: dut (-8.85)      64.1     78.3     76.3     69.8     90.1     cir (72.50)
57    40: ens (-8.97)      71.8     40.9     98.7     69.6     43.5     ntw (72.00)
58    41: ntw (-11.15)     81.5     79.8     66.6     25.5     67.6     anu (70.57)
59    42: anu (-11.50)     47.2     73.0     92.2     90.0     48.1     rcu (69.79)
60    43: tub (-12.20)     66.2     82.4     71.0     55.4     99.9     mel (69.67)
61    44: mel (-23.98)     56.1     70.2     83.7     83.3     50.4     lms (68.38)

```

62	45: lms (-25.43)	81.5	68.1	61.0	31.1	87.8	ens (68.35)
63	46: bro (-27.18)	58.5	54.9	96.8	52.3	38.6	wtu (67.86)
64	47: frei (-34.42)	54.2	51.6	89.5	49.7	99.9	tech (67.06)
65	48: wtu (-35.05)	61.8	73.5	73.7	51.9	62.2	bro (66.49)
66	49: tech (-37.95)	54.9	71.0	85.1	51.7	40.1	man (66.33)
67	50: itmo (-38.50)	58.0	32.0	98.7	39.2	68.7	zhej (65.34)
68	51: zhej (-43.70)	73.5	70.4	60.7	22.6	75.7	frei (65.08)
69	52: man (-44.83)	63.5	71.9	62.9	84.1	42.1	unsw (63.65)
70	53: kuj (-47.40)	75.4	72.8	49.5	28.3	51.4	kuj (62.77)
71	54: kul (-49.98)	35.2	55.8	92.0	46.0	88.3	sou (62.15)
72	55: unsw (-54.88)	60.2	58.2	70.5	87.0	44.3	shJi (61.35)
73	56: glas (-56.98)	35.2	52.5	91.2	85.8	39.2	itmo (60.52)
74	57: utw (-59.27)	38.2	52.8	87.0	69.0	60.0	kul (60.47)
75	58: unlu (-60.08)	35.2	44.2	87.4	99.7	54.1	glas (59.78)
76	59: naji (-60.52)	51.4	76.9	48.8	39.7	74.4	utw (59.40)
77	60: sou (-60.83)	48.2	60.7	75.5	87.4	43.2	stut (58.85)
78	61: hkpu (-62.05)	46.8	36.5	91.4	73.2	41.5	naji (58.61)
79	62: qut (-66.17)	45.5	42.6	82.8	75.2	63.0	tud (58.28)
80	63: humb (-68.10)	48.4	31.3	94.7	41.5	45.5	unlu (58.04)
81	64: shJi (-69.72)	66.9	68.3	62.4	22.8	38.5	qut (57.99)
82	65: stut (-69.90)	54.2	60.6	61.1	36.3	97.8	hkpu (57.69)
83	66: tud (-70.83)	46.6	53.6	75.9	53.7	66.5	albt (57.63)
84	67: tlavu (-71.50)	34.1	57.2	89.0	45.3	38.6	mil (57.47)
85	68: cihk (-72.20)	42.4	44.9	80.1	76.2	67.9	hels (57.40)
86	69: albt (-72.33)	39.2	53.3	69.9	91.9	75.4	cihk (57.33)
87	70: indis (-72.53)	56.9	76.1	49.3	20.1	41.5	tlavu (57.19)
88	71: ariz (-75.10)	28.4	61.8	84.3	59.3	42.0	indis (57.04)
89	72: kth (-77.10)	44.8	42.0	83.6	71.6	39.2	ariz (56.79)
90	73: hels (-79.55)	48.8	49.6	80.4	50.6	39.5	kth (56.36)
91	74: eind (-82.85)	32.4	48.4	81.5	72.2	45.8	humb (55.34)
92	75: mil (-83.67)	46.4	64.3	69.2	44.1	38.5	eind (54.36)

The first inversion we observe in Listing 13.11 (Lines 20-21) 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 53). 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 80).

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

$$r(x \text{ is first-ranked}) = \sum_{y \neq x} r((x \succsim y) + (y \not\succsim x)) = \sum_{y \neq x} (r(x \succsim y) - r(y \succsim x)). \quad (13.1)$$

Using the robust outranking characteristics of digraph rdg , we may thus explicitly compute, for instance, ETH Zürich's 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] - rdg.relation[y][x])
6 >>> print(x, nfx)
7     ethz 116.950

```

In Listing 13.11 (Line 18), we may now verify that ETH Zürich obtains indeed the highest NETFLOWS score, and gives, hence the *most credible* first-ranked CS Dept of the 75 potential candidates.

How may we now convince the reader, that our pairwise outranking based ranking result here appears more objective and trustworthy, than the classic value theory based THE ranking by overall scores?

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 may thus conveniently be checked. The ETH Zürich CS Dept is, for instance, first ranked before Caltech's Dept in both previous rankings. Lest us check the preferential reasons.

Listing 13.12 Comparing pairwise criteria performances

```

1 >>> rdg.showPairwiseOutrankings('ethz','calt')
2 ----- pairwise comparisons -----
3 Valuation in range: -100.00 to +100.00
4 Comparing actions : ('ethz', 'calt')
5 crit. wght. g(x) g(y) diff | ind pref r() |
6 -----
7 'gcit' 27.50 97.10 99.80 -2.70 | 2.50 5.00 +0.00 |
8 'gind' 5.00 64.10 85.90 -21.80 | 2.50 5.00 -5.00 |
9 'gint' 7.50 93.60 59.10 +34.50 | 2.50 5.00 +7.50 |
10 'gres' 30.00 97.30 96.00 +1.30 | 2.50 5.00 +30.00 |
11 'gtch' 30.00 89.20 91.50 -2.30 | 2.50 5.00 +30.00 |
12                                     r(x >= y): +62.50
13 crit. wght. g(y) g(x) diff | ind pref r() |
14 -----
15 'gcit' 27.50 99.80 97.10 +2.70 | 2.50 5.00 +27.50 |
16 'gind' 5.00 85.90 64.10 +21.80 | 2.50 5.00 +5.00 |
17 'gint' 7.50 59.10 93.60 -34.50 | 2.50 5.00 -7.50 |
18 'gres' 30.00 96.00 97.30 -1.30 | 2.50 5.00 +30.00 |
19 'gtch' 30.00 91.50 89.20 +2.30 | 2.50 5.00 +30.00 |
20                                     r(y >= x): +85.00

```

A significant positive performance difference (+34.50), concerning the *International outlook* criterion (of 7.5% significance), may be observed 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),

may be observed, this time, in favour of the Caltech Dept. 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 ETH Zürich first, before Caltech. Yet, the slightly positive performance difference (+2.70) between Caltech and ETH Zürich on the *Citations* criterion (of 27.5% significance) confirms an *at least as good as* situation in favour of the Caltech Dept.

The inverse negative performance difference (-2.70), however, is neither *significant* (< -5.00), nor *insignificant* (> -2.50), and does hence neither confirm nor infirm a *not at least as good as* situation in disfavour of ETH Zürich. We observe here a convincing argument of 27.5% significance for putting Caltech first, before ETH Zürich.

Notice finally, that, on the *Teaching* and *Research* criteria of total significance 60%, both Depts do, with performance differences $< \text{abs}(2.50)$, one as well as the other. As these two major performance criteria necessarily support together always the highest significance with the imposed significance weight preorder: 'gtch' = 'gres' \wedge 'gcit' \wedge 'gint' \wedge 'gind', both outranking situations get in fact globally confirmed at stability level +2 (see Chapter 19).

We may well illustrate all such stable outranking situations with a browser view of the corresponding robust relation map using our NETFLOWS ranking.

```
1 >>> rdg.showHTMLRelationMap(tableTitle='Robust Outranking Map',
2 ...                                rankingRule='NetFlows')
```

In Fig. 13.2, *dark green*, resp. *light green* marked positions show certainly, resp. positively valid outranking situations, whereas *dark red*, resp. *light red* marked positions show certainly, respectively positively valid outranked situations. In the left upper corner we may verify that the five top-ranked Depts ([ethz', 'calt', 'oxf', 'mit', 'cmel']) are indeed mutually outranking each other and thus are to be considered all indifferent. They are even robust CONDORCET winners, i.e positively outranking all other Depts. We may by the way notice that no certainly valid outranking (dark green) and no certainly valid outranked situations (dark red) appear below, resp. above the principal diagonal; none of these are hence violated by our NETFLOWS ranking.

The non reflexive *white* positions in the relation map, mark outranking or outranked situations that are not robust with respect to the given significance weight preorder. They are, hence, put into doubt and 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 may be formally evaluated Footnote[27].

Listing 13.13 Measuring the quality of the NETFLOWS ranking result

```
1 >>> corrnf = rdg.computeRankingCorrelation(nfRanking)
2 >>> rdg.showCorrelation(corrnf)
3 Correlation indexes:
4     Crisp ordinal correlation : +0.901
5     Epistemic determination   : 0.563
6     Bipolar-valued equivalence : +0.507
```

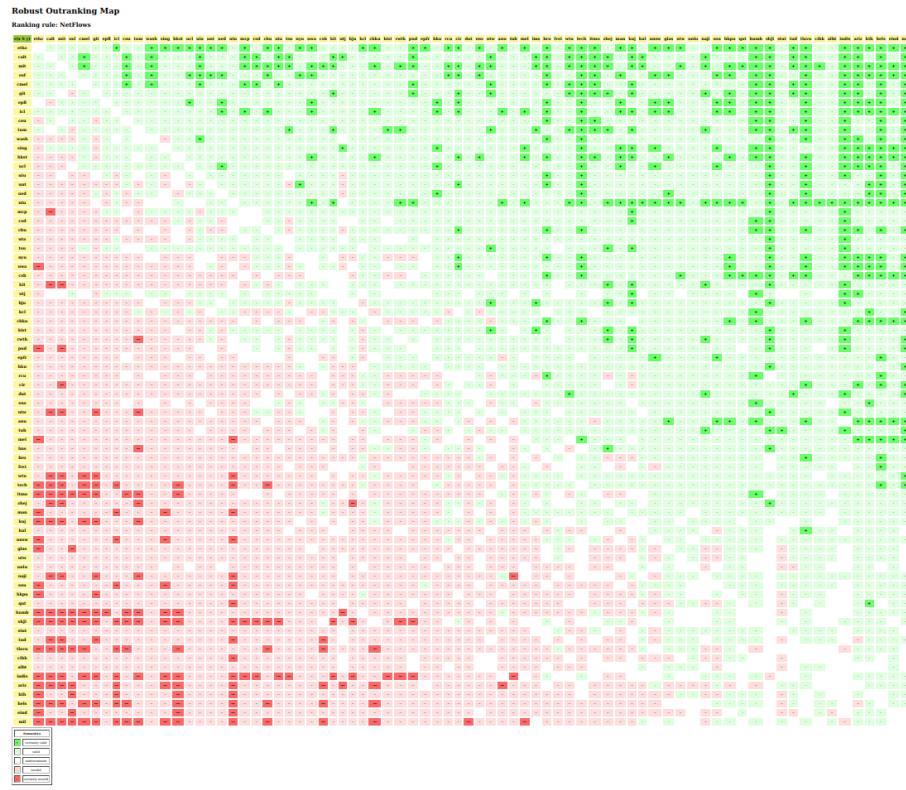


Fig. 13.2 Relation map of the robust outranking relation

In Listing 13.13 (Line 4), we may notice that the NETFLOWS ranking result is indeed highly ordinally correlated (+0.901, in KENDALL's tau index sense) with the pairwise global robust outranking relation. Their bipolar-valued *relational equivalence* value (+0.51, Line 6) indicates a more than 75% criteria significance support.

We may as well check how the NETFLOWS ranking rule is actually balancing the five ranking criteria.

```

1 >>> rdg.showRankingConsensusQuality(nfRanking)
2 Criterion (weight): correlation
3 -----
4     gtch (0.300): +0.660
5     gres (0.300): +0.638
6     gcit (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 correlations with the marginal performance criterion rankings are nearly respecting the given significance weights preorder: 'gtch' = 'gres' < 'gcit' < 'gint' < 'gind' (see Lines 4-8 above). The mean marginal correlation is quite high (+0.51). Coupled with a low standard deviation (0.187), we obtain a rather fairly balanced ranking result (Lines 10-12).

We may also inspect the mutual correlation indexes observed between the marginal criterion robust outranking relations.

```

1 >>> rdg.showCriteriaCorrelationTable()
2   Criteria ordinal correlation index
3   | gcit      gind      gint      gres      gtch
4   -----|-----
5   gcit | +1.00    -0.11    +0.24    +0.13    +0.17
6   gind |           +1.00    -0.18    +0.15    +0.15
7   gint |           -        +1.00    +0.04    -0.00
8   gres |           -        -        +1.00    +0.67
9   gtch |           -        -        -        +1.00

```

Slightly contradictory (-0.11) appear the *Citations* and *Industrial income* criteria (Line 5 Column 3). 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).

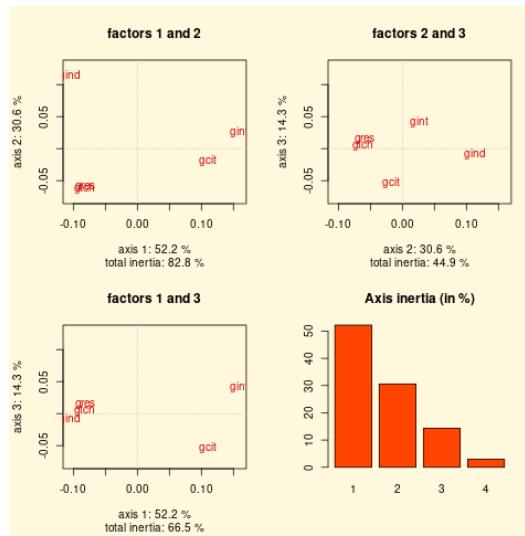
A Principal Component Analysis may well illustrate the previous findings.

```

1 >>> rdg.export3DplotOfCriteriaCorrelation(graphType='png')

```

Fig. 13.3 3D PCA plot of the pairwise criteria correlation table. 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 (see Fig. 13.3), 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 may, finally, be interesting to assess, similarly, the ordinal correlation of the THE overall scores based ranking with respect to our robust outranking situations.

Listing 13.14 Computing the ordinal quality of the THE ranking

```

1 >>> # theScores = [(xScore_1,x_1), (xScore_2,x_2),... ]
2 >>> # is sorted in decreasing order of xscores
3 >>> theRanking = [item[1] for item in theScores]
4 >>> corrthe = rdg.computeRankingCorrelation(theRanking)
5 >>> rdg.showCorrelation(corrthe)
6     Correlation indexes:
7         Crisp ordinal correlation : +0.907
8         Epistemic determination : 0.563
9         Bipolar-valued equivalence : +0.511
10 >>> rdg.showRankingConsensusQuality(theRanking)
11     Criterion (weight): correlation
12     -----
13     gtch (0.300): +0.683
14     gres (0.300): +0.670
15     gcit (0.275): +0.319
16     gint (0.075): +0.161
17     gind (0.050): +0.106
18     Summary:
19     Weighted mean marginal correlation (a): +0.511
20     Standard deviation (b) : +0.210
21     Ranking fairness (a)-(b) : +0.302

```

The THE ranking result is similarly correlated (+0.907, Line 7 in Listing 13.3) with the pairwise global robust outranking relation. By its overall weighted scoring rule, the THE ranking induces marginal criterion correlations that are naturally 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 NET-FLOWS ranking. Yet, its standard deviation is higher, which leads to a slightly less fair balancing of the three major ranking criteria.

To conclude, let us emphasize, that, without any commensurability hypothesis and by taking, furthermore, into account, first, the always present more or less imprecision of any performance grading and, secondly, solely ordinal criteria significance weights, we may obtain here with our robust outranking approach a very similar ranking result with more or less a same, when not better, preference modelling quality. A convincing heatmap view of the 25 first-ranked Institutions may be generated in the default system browser with following command.

```

1 >>> rdg.showHTMLPerformanceHeatmap(
2     ...           WithActionNames=True,\ 
3     ...           outrankingModel='this',\ 
4     ...           rankingRule='NetFlows', \

```

```

5     ...          ndigits=1, \
6     ...          Correlations=True, \
7     ...          fromIndex=0, toIndex=25)

```

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
California 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-Champaign (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

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 equisignificant. Curious to see the ranking results under such settings.

Chapter 14

The best students, where do they study? A rating case study

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 Footnote[28]. 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. Footnote[29]. The eventually published results by the *Spiegel* magazine concerned nearly 50,000 students, enroled in one of fifteen popular academic subjects, like *German Studies, Life Sciences, Psychology, Law or CS*. 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 Footnote[29]. Based on this published data Footnote[28], we present and discuss in this chapter, how to *rate* with the help of our DIGRAPH3 software ressources the apparent global *enrolment quality* of these 41 higher education institutions.

14.1 The performance tableau

Published data of the 2004 *Spiegel* student survey is stored, for our evaluation purpose here, in a file named *studentenSpiegel04.py* of *PerformanceTableau* format Footnote[32].

Listing 14.1 The 2004 Spiegel students survey data

```
1 >>> from perfTabs import PerformanceTableau
2 >>> t = PerformanceTableau('studentenSpiegel04')
3 >>> t
4     ----- PerformanceTableau instance description -*-
5     Instance class      : PerformanceTableau
```

```
6     Instance name      : studentenSpiegel04
7     Actions            : 41 (Universities)
8     Criteria           : 15 (academic subjects)
9     NA proportion (%) : 27.3
10    Attributes         : ['name', 'actions',
11                           'objectives', 'criteria',
12                           'weightPreorder',
13                           'evaluation']
14 >>> t.showHTMLPerformanceHeatmap(ndigits=1,\n15     rankingRule=None)
15 ...
```

criteria	germ	pol	psy	soc	law	eco	mgt	bio	med	phys	chem	math	info	elec	mec	
weights	+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	+1.00	+1.00	
aach	53.3	50.8	62.7	51.0	NA	NA	49.6	52.2	49.5	59.1	53.6	58.6	54.6	57.2	54.4	
aug	57.9	54.3	NA	54.8	45.6	NA	54.3	NA	NA	62.3	NA	61.2	58.1	NA	NA	
berf	54.7	61.4	59.8	55.5	45.7	50.5	52.2	51.6	49.0	61.6	57.4	NA	54.9	NA	NA	
berh	57.3	58.5	59.8	59.2	48.8	59.5	55.5	55.0	52.3	61.9	53.2	57.9	55.8	NA	NA	
bertu	51.4	NA	57.7	59.1	NA	49.6	54.0	NA	NA	58.9	52.0	56.8	55.4	56.1	54.3	
bie	51.4	NA	54.4	55.6	41.9	NA	50.7	49.7	NA	53.9	54.2	56.3	55.8	NA	NA	
boc	53.9	NA	55.2	NA	39.1	NA	NA	48.0	49.8	56.8	53.3	57.6	NA	54.2	54.4	
bon	54.1	57.3	60.3	56.0	47.2	53.6	NA	50.1	3.0	59.9	53.1	59.4	53.7	NA	NA	
brau	53.5	54.0	NA	51.5	NA	NA	53.4	53.1	NA	59.8	50.1	54.7	52.6	54.5	55.2	
brem	56.9	55.5	52.5	54.5	40.9	NA	55.4	53.3	NA	59.7	NA	NA	54.1	50.1	NA	
chem	54.3	57.1	60.8	53.3	NA	NA	52.7	NA	NA	NA	NA	NA	57.7	57.5	53.6	
darm	1.0	59.7	58.6	52.0	NA	NA	NA	1.0	NA	62.5	2.0	59.4	3.0	NA	56.1	
dres	55.2	55.9	60.6	56.2	44.0	56.7	54.8	55.3	49.2	59.9	55.8	57.8	56.2	56.1	54.8	
dsd	53.5	NA	57.5	48.8	44.9	NA	50.5	47.3	50.5	NA	53.5	NA	NA	NA	NA	
duis	50.6	52.5	NA	47.9	NA	NA	47.5	NA	48.0	54.6	52.8	51.6	56.8	53.6	51.9	
erl	57.9	55.1	58.7	55.4	42.9	NA	55.6	51.8	49.3	60.3	54.0	60.5	54.6	55.9	55.1	
fran	51.7	53.1	58.0	51.5	41.9	53.5	52.0	51.3	51.2	62.1	55.5	57.0	52.4	NA	NA	
frei	57.8	60.5	64.1	57.5	50.7	53.3	NA	55.4	54.2	61.6	57.0	60.6	58.1	NA	NA	
gie	53.0	59.0	58.0	NA	41.9	NA	51.2	50.4	50.0	57.6	NA	NA	NA	NA	NA	
goet	58.7	56.3	59.8	53.5	44.8	53.6	52.6	50.5	48.9	60.4	53.9	63.1	NA	NA	NA	
ham	57.0	60.2	57.3	53.6	44.1	52.1	49.8	52.7	49.2	56.4	54.2	54.9	54.7	NA	NA	
han	50.4	52.8	NA	49.9	41.2	NA	NA	53.8	NA	57.5	53.6	56.6	58.8	53.5	53.6	
hei	61.4	59.5	59.8	52.2	51.1	54.4	NA	55.2	55.5	60.9	56.7	61.3	NA	NA	NA	
jena	56.5	55.8	58.5	52.8	45.3	NA	56.2	52.7	51.1	61.6	57.8	NA	57.2	NA	NA	
kiel	51.9	52.2	58.4	NA	45.1	50.5	52.8	52.7	49.6	59.7	52.8	NA	54.9	54.2	NA	
koel	51.7	57.6	58.9	56.1	46.1	56.1	54.6	51.8	50.7	58.7	54.0	56.5	NA	NA	NA	
kons	56.9	65.9	59.1	54.7	48.3	59.0	NA	55.7	NA	59.9	58.0	NA	59.7	NA	NA	
ksl	NA	55.9	50.8	NA	59.7	54.6	62.2	56.2	57.5	56.3						
leip	57.4	60.4	62.4	59.5	46.3	NA	54.6	56.9	51.5	62.2	57.0	NA	3.0	NA	NA	
main	54.2	57.9	56.9	55.7	46.5	50.7	53.1	50.0	49.2	60.8	56.3	54.7	NA	NA	NA	
marb	53.6	54.7	57.6	59.8	40.3	NA	55.5	53.2	51.1	62.8	57.8	NA	55.6	NA	NA	
mnh	52.2	57.2	61.1	55.0	45.0	57.0	59.7	NA	NA	NA	NA	NA	NA	58.6	NA	NA
mnst	55.4	56.7	62.2	56.3	46.4	54.1	56.3	51.2	52.8	55.2	55.0	56.9	57.7	NA	NA	
mu	57.2	60.1	60.9	54.0	47.3	55.8	57.5	50.4	52.6	62.0	58.1	59.6	57.1	NA	NA	
reg	54.8	55.4	62.1	NA	46.1	52.3	55.5	54.6	50.9	60.5	55.8	59.2	NA	NA	NA	
saar	57.9	NA	56.5	NA	48.1	NA	52.2	NA	49.6	61.2	56.2	NA	57.6	NA	NA	
stu	52.5	58.4	NA	NA	NA	NA	56.6	57.1	NA	61.5	55.2	60.6	59.8	60.2	57.8	
tri	54.1	58.0	58.3	54.9	46.3	52.8	52.8	NA	NA	NA	NA	60.7	52.3	NA	NA	
tueb	57.9	57.7	58.4	1.0	46.8	60.8	54.4	53.7	52.1	61.6	57.5	NA	55.2	NA	NA	
tum	NA	NA	NA	NA	NA	NA	68.0	53.6	60.1	62.8	58.8	62.6	58.2	58.2	56.9	
wrzb	56.9	56.0	59.8	NA	46.4	53.3	52.8	53.0	52.2	60.2	56.6	NA	55.9	NA	NA	

Color legend:

quantile	14.29%	28.57%	42.86%	57.14%	71.43%	85.71%	100.00%
----------	--------	--------	--------	--------	--------	--------	---------

Fig. 14.1 Average quality of enroled students per academic subject

In Fig. 14.1, the fifteen popular academic subjects are grouped into topical 'Faculties': - *Humanities*; - *Law, Economics & Management*; - *Life Sciences & Medicine*; - *Natural Sciences & Mathematics*; and - *Technology*. All fifteen subjects are considered equally significant for our evaluation problem (see Row 2). The recorded average enrolment quality scores appear coloured along a 7-tiling scheme per subject (see last Row).

We may by the way notice that TU Dresden is the only Institution showing enrolment quality scores in all the fifteen academic subjects. Whereas, on the one side, TU Muenchen and Kaiserslautern are only valued in *Sciences* and *Technology* subjects. On the other side, Mannheim, is only valued in *Humanities* and *Law, Economics & Management* studies. Most of the 41 Universities are not valued in *Engineering* studies. We are, hence, facing a large part (27.3%) of irreducible missing data (see Listing 14.1 Line 9 and Chapter 16 coping with missing data).

Details of the enrolment quality criteria (the academic subjects) may be consulted in a browser view (see :numref:`spiegelCriteria` below).

```
>>> t.showHTMLCriteria()
```

#	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 Details of the rating criteria

The evaluation of the individual quality score for a participating student actually depends on his or her mainly enroled subject Footnote[29]. The apparent quality measurement scales thus largely differ indeed from subject to subject (see Fig. 14.2), like *Law Studies* (35.0 – 65 – 0) and *Politology* (50.0 – 70.0). The recorded average enrolment quality scores, hence, are in fact *incommensurable* between the subjects.

To take furthermore into account a potential and very likely imprecision of the individual quality scores' computation, we shall assume that, for all subjects, an average enrolment quality score difference of 0.1 is *insignificant*, wheras a difference of 0.5 is sufficient to positively attest a *better* enrolment quality.

The apparent incommensurability and very likely imprecision of the recorded average enrolment quality scores, renders meaningless any global averaging over the subjects per University of the enrolment quality. We shall therefore, similarly

to the methodological approach of the *Spiegel* authors Footnote[29], proceed with an *order statistics* based rating-by-ranking approach (see Chapter 10 on rating with learned quantile norms).

14.2 Rating-by-ranking with lower-closed quantile limits

The *Spiegel* authors opted indeed for a simple 3-tiling of the Universities per valued academic subject, followed by an average BORDA scores based global ranking Footnote[29]. Here, our epistemic logic based outranking approach, allows us, with adequate choices of *indifference* (0.1) and *preference* (0.5) discrimination thresholds, to estimate *lower-closed* 9-tiles of the enrolment quality scores per subject and rank conjointly, with the help of the COPELAND ranking rule Footnote[34] applied to a corresponding bipolar-valued outranking digraph, the 41 Universities **and** the lower limits of the estimated 9-tiles limits.

First, we therefore need, with the help of the `PerformanceQuantiles` constructor, to estimate the lowerclosed 9-tiling of the average enrolment quality scores per academic subject.

Listing 14.2 Computing 9-tiles of the enrolment quality scores per subject

```

1 >>> from performanceQuantiles import PerformanceQuantiles
2 >>> pq = PerformanceQuantiles(t,numberOfBins=9,LowerClosed=True)
3 >>> pq
4     *----- PerformanceQuantiles instance description -----*
5     Instance class    : PerformanceQuantiles
6     Instance name     : 9-tiled_performances
7     Criteria          : 15
8     Quantiles         : 9 (LowerClosed)
9     History sizes    :
10    'germ': 39, 'pol': 34, 'psy': 34, 'soc': 32,
11    'law': 32, 'eco': 21, 'mgt': 34,
12    'bio': 34, 'med': 28, 'phys': 37, 'chem': 35, 'math': 27,
13    'info': 33, 'elec': 14, 'mec': 13,}
```

The history sizes, reported in Listing 14.2 above, indicate the number of Universities valued in each one of the popular fifteen subjects. *German Studies*, for instance, are valued for 39 out of 41 Universities, whereas *Electrical and Mechanical Engineering* are only valued for 14, respectively 13 Institutions. None of the fifteen subjects are valued in all the 41 Universities Footnoe[30].

We may inspect the resulting 9-tiling limits in a browser view.

```

1 >>> pq.showHTMLLimitingQuantiles(Transposed=True,Sorted=False,\n2 ...           ndigits=1,title='9-tiled quality score limits')
```

We add, now, these 9-tiling quality score limits to the enrolment quality records of the 41 Universities and rank all these records conjointly together with the help of the `LearnedQuantilesRatingDigraph` constructor and by using the COPELAND ranking rule.

Fig. 14.3 9-tiling quality score limits per academic subject. 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). Universities evaluated in *Law studies* but not in *Politology*, like the University of Bielefeld, would see their enrolment quality unfairly weakened when simply averaging the enrolment quality scores over valued subjects

9-tiled quality score limits

Sampling sizes between 13 and 39.

criterion	0.00	0.11	0.22	0.33	0.44	0.56	0.67	0.78	0.89	1.00
bio	45.0	49.9	50.5	51.4	52.3	53.0	53.5	54.8	55.5	57.1
chem	45.0	52.8	53.5	54.0	54.4	55.6	56.4	57.1	57.8	58.8
eco	49.6	50.6	52.2	53.3	53.5	53.9	55.8	56.8	59.3	60.8
elec	50.1	53.6	54.2	54.4	55.9	56.1	57.3	57.5	59.1	60.2
germ	45.0	51.5	52.4	53.5	54.1	55.1	56.9	57.3	57.9	61.4
info	45.0	52.5	54.6	54.9	55.7	56.2	57.2	58.0	58.7	59.8
law	39.1	41.6	43.0	44.9	45.4	46.1	46.4	47.2	48.5	51.1
math	51.6	54.9	56.6	57.0	57.9	59.4	60.5	60.7	62.2	63.1
mec	51.9	53.6	54.2	54.4	54.7	55.1	55.8	56.4	57.4	57.8
med	45.0	49.0	49.2	49.6	50.2	51.0	51.4	52.3	54.0	60.1
mgt	47.5	50.7	52.2	52.8	53.5	54.6	55.5	55.7	56.8	68.0
phys	53.9	56.9	58.9	59.7	60.0	60.7	61.6	61.8	62.3	62.8
pol	50.8	53.0	54.9	55.8	56.7	57.6	58.3	59.6	60.4	65.9
psy	52.5	56.8	57.7	58.3	58.6	59.7	59.8	60.8	62.2	64.1
soc	45.0	50.5	52.0	53.4	54.5	55.0	55.6	56.2	59.1	59.8

```

1 >>> from sortingDigraphs import \
2 ...                   LearnedQuantilesRatingDigraph
3 >>> lqr = LearnedQuantilesRatingDigraph(pq,t,\ 
4 ...                                         rankingRule='Copeland')
```

The resulting ranking of the 41 Universities including the lower-closed 9-tiling score limits may be nicely illustrated with the help of a corresponding heatmap view

```

1 >>> lqr.showHTMLRatingHeatmap(colorLevels=7,\ 
2 ...                           Correlations=True,\ 
3 ...                           ndigits=1,rankingRule='Copeland')
```

The ordinal correlation (+0.967) Footnote[35] of the COPELAND ranking with the underlying bipolar-valued outranking digraph is very high (see Fig. 14.4 Row 1). Most correlated subjects with this rating-by-ranking result appear to be *German Studies* (+0.51), *Chemistry* (+0.48), *Management* (+0.47) and *Physics* (+0.46). Both *Electrical* (+0.07) and *Mechanical Engineering* (+0.05) are the less correlated subjects (see Row 3).

From the actual ranking position of the lower 9-tiling limits, we may now immediately deduce the 9-tile enrolment quality equivalence classes. No University reaches the highest 9-tile ([0.89 – 1]). In the lowest 9-tile ([0.00 – 0.11]) we find the University Duisburg. The complete rating result may be easily printed out as follows.

Listing 14.3 Computing 9-tiles of the enrolment quality scores per subject

```

1 >>> lqr.showQuantilesRating()
2      ----- Quantiles rating result -----
3      [0.89 - 1.00] []
4      [0.78 - 0.89[ ['tum','frei','kons','leip','mu','hei']
5      [0.67 - 0.78[ ['stu','berh']
```

```

6 [0.56 - 0.67[ ['aug','mnh','tueb','mnst','jena',
7           'reg','saar']
8 [0.44 - 0.56[ ['wrzb','dres','ksl','marb','berf',
9           'chem','koel','erl','tri']
10 [0.33 - 0.44[ ['goet','main','bon','brem']
11 [0.22 - 0.33[ ['fran','ham','kiel','aach',
12           'bertu','brau','darm']
13 [0.11 - 0.22[ ['gie','dsd','bie','boc','han']
14 [0.00 - 0.11[ ['duis']

```

Following Universities: TU München, Freiburg, Konstanz, Leipzig, München as well as Heidelberg, appear best rated in the eighth 9-tile ([0.78 – 0.89[, see Listing 14.3 Line 4). Lowest-rated in the first 9-tile, as mentioned before, appears University Duisburg (Line 14). Midfield, the fifth 9-tile ([0.44 – 0.56[), consists of the Universities Würzburg, TU Dresden, Kaiserslautern, Marburg, FU Berlin, Chemnitz, Köln , Erlangen-Nürnberg and Trier (Lines 8-9).

A corresponding *graphviz* drawing may well illustrate all these enrolment quality equivalence classes.

```

1 >>> lqr.exportRatingByRankingGraphViz(fileName='ratingResult', \
2 ...                                     graphSize='12,12')
3     *---- exporting a dot file for GraphViz tools -----*
4     Exporting to ratingResult.dot
5     dot -Grankdir=TB -Tpdf dot -o ratingResult.png

```

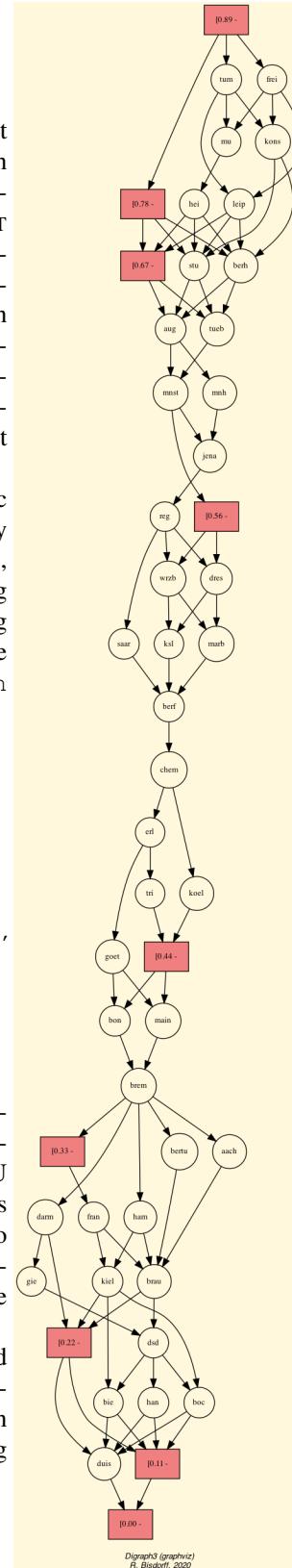
Fig. 14.6 Fused Copeland and NetFlows ratings
 We have noticed in Chapter 8 [8], that there is not a single optimal rule for ranking from a given outranking digraph. The COPELAND rule, for instance, has the advantage of being CONDORCET consistent, i.e. when the outranking digraph models in fact a linear 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 many circuits, all potential ranking rules may give very divergent ranking results, and hence also substantially divergent rating-by-ranking results.

It is, hence, interesting, to verify if the epistemic fusion of the rating-by-ranking results, one may obtain when applying two different ranking rules, like the COPELAND and the NETFLOWS ranking rule, does actually confirm our rating-by-ranking result shown in Fig. 14.5. For this purpose we make usage of the RankingsFusionDigraph constructor (see Line 9 below).

```
>>> lqr = LearnedQuantilesRatingDigraph(\n...     pq,t,rankingRule='Copeland')\n>>> lqr1 = LearnedQuantilesRatingDigraph(\n...     pq,t,rankingRule='NetFlows')\n>>> from transitiveDiGraphs import\\n\n...     RankingsFusionDigraph\n>>> rankings = [lqr.actionsRanking,\n...     lqr1.actionsRanking]\n>>> rf = RankingsFusionDigraph(lqr,rankings)\n>>> rf.exportGraphViz(fileName='fusionResult',\n...     WithRatingDecoration=True,\n...     graphSize='30,30')\n  exporting a dot file for GraphViz tools\n  Exporting to fusionResult.dot\n  dot -Grankdir=TB -Tpng fusionResult.dot\\n\n  ...
```

In the Figure here, we notice that many Universities appear now rated into several adjacent 9-tiles. The previously best-rated Universities: TU München, Freiburg, München, Leipzig, as well as Heidelberg, for instance, appear now sorted into the 7th and 8th 9-tile ([0.67 – 0.89]), whereas Konstanz is now, even more imprecisely, rated into the 6th, the 7th and the 8th 9-tile.

How confident, hence, is our precise Copeland rating-by-ranking result? To investigate this question, let us now inspect the outranking digraph on which we actually apply the COPELAND ranking rule.



14.3 Inspecting the bipolar-valued outranking digraph

We say that University x *outranks* (resp. *is outranked by*) University y in enrolment quality when there exists a majority (resp. only a minority) of valued subjects showing an *at least as good as* average enrolment quality score. To compute these outranking situations, we use the `BipolarOutrankingDigraph` constructor.

Listing 14.4 Computing 9-tiles of the enrolment quality scores per subject

```

1 >>> from outrankingDigraphs import BipolarOutrankingDigraph
2 >>> dg = BipolarOutrankingDigraph(t)
3 >>> dg
4     *----- Object instance description -----*
5     Instance class      : BipolarOutrankingDigraph
6     Instance name       : rel_studentenSpiegel04
7     Actions             : 41 (Universities)
8     Criteria            : 15 (subjects)
9     Size                : 828 (outranking situations)
10    Determinateness (%) : 63.67
11    Valuation domain   : [-1.00;1.00]
12 >>> dg.computeTransitivityDegree(Comments=True)
13     Transitivity degree of digraph <rel_studentenSpiegel04>:
14     triples x>y>z: 57837, closed: 30714, open: 27123
15     (closed/triples) = 0.531
16 >>> dg.computeSymmetryDegree(Comments=True)
17     Symmetry degree of digraph <rel_studentenSpiegel04>:
18     arcs x>y: 793, symmetric: 35, asymmetric: 758
19     symmetric/arcs = 0.044

```

The bipolar-valued outranking digraph dg (see Listing 14.4 Line 2), obtained with the given performance tableau t , shows 828 positively validated pairwise outranking situations (Line 9). Unfortunately, the transitivity of digraph dg is far from being satisfied: nearly half of the transitive closure is missing (Line 15). Despite the rather large preference discrimination threshold (0.5) we have assumed (see Fig. 14.2), there does not occur many indifference situations (Line 19).

We furthermore check if there exists any *cyclic* outranking situations.

Listing 14.5 Enumerating chordless outranking circuits

```

1 >>> from outrankingDigraphs import BipolarOutrankingDigraph
2 >>> dg.computeChordlessCircuits()
3 >>> dg.showChordlessCircuits()
4     *---- Chordless circuits ----*
5     93 circuits.
6     1: ['aach','bie','darm','brau'] ,credibility : 0.067
7     2: ['aach','bertu','brau'] ,credibility : 0.200
8     3: ['aach','bertu','brem'] ,credibility : 0.067
9     4: ['aach','bertu','ham'] ,credibility : 0.200
10    5: ['aug','tri','marb'] ,credibility : 0.067
11    6: ['aug','jena','marb'] ,credibility : 0.067
12    7: ['aug','jena','koel'] ,credibility : 0.067
13    ...
14    ...

```

```

15    29: ['berh','kons','mu'] ,credibility : 0.133
16    ...
17    ...
18    88: ['main','mnh','marb'] ,credibility : 0.067
19    89: ['marb','saar','wrzb'] ,credibility : 0.067
20    90: ['marb','saar','reg'] ,credibility : 0.067
21    91: ['marb','saar','mnst'] ,credibility : 0.133
22    92: ['marb','saar','tri'] ,credibility : 0.067
23    93: ['mnh','mu','stu'] ,credibility : 0.133

```

Here we observe indeed 93 such outranking circuits, like: Berlin Humboldt > Konstanz > München > Berlin Humboldt, supported by a $(0.133 + 1.0)/2 = 56.7\%$ majority of subjects Footnote[31] (see circuit 29 above). In the COPELAND ranking result shown in Fig. 14.4, these Universities appear positioned respectively at ranks 10, 4 and 6. In the NETFLOWS ranking result they would appear respectively at ranks 10, 6 and 5, thus inverting the positions of Konstanz and München. The occurrence in digraph dg of so many outranking circuits makes thus *doubtful* any *forced* linear ranking, independently of the specific ranking rule we might have applied.

To effectively check the quality of our COPELAND rating-by-ranking result, we shall now compute a direct sorting into 9-tiles of the enrolment quality scores, without using any outranking digraph based ranking rule.

14.4 Rating by quantiles sorting

In our case here, the Universities represent the decision actions: *where to study*. We say now that University x is sorted into the lower-closed 9-tile q when the performance record of x positively outranks the lower limit record of 9-tile q and x does not positively outrank the upper limit record of 9-tile q .

Listing 14.6 Enumerating chordless outranking circuits

```

1 >>> lqr.showActionsSortingResult()
2   Quantiles sorting result per decision action
3   [0.33 - 0.44[: aach with credibility: 0.13 = min(0.13,0.27)
4   [0.56 - 0.89[: aug with credibility: 0.13 = min(0.13,0.27)
5   [0.44 - 0.67[: berf with credibility: 0.13 = min(0.13,0.20)
6   [0.78 - 0.89[: berh with credibility: 0.13 = min(0.13,0.33)
7   [0.22 - 0.44[: bertu with credibility: 0.20 = min(0.33,0.20)
8   [0.11 - 0.22[: bie with credibility: 0.20 = min(0.33,0.20)
9   [0.22 - 0.33[: boc with credibility: 0.07 = min(0.07,0.07)
10  [0.44 - 0.56[: bon with credibility: 0.13 = min(0.20,0.13)
11  [0.33 - 0.44[: brau with credibility: 0.07 = min(0.07,0.27)
12  [0.33 - 0.44[: brem with credibility: 0.07 = min(0.07,0.07)
13  [0.44 - 0.56[: chem with credibility: 0.07 = min(0.13,0.07)
14  [0.22 - 0.56[: darm with credibility: 0.13 = min(0.13,0.13)
15  [0.56 - 0.67[: dres with credibility: 0.27 = min(0.27,0.47)
16  [0.22 - 0.33[: dsd with credibility: 0.07 = min(0.07,0.07)
17  [0.00 - 0.11[: duis with credibility: 0.33 = min(0.73,0.33)
18  [0.44 - 0.56[: erl with credibility: 0.13 = min(0.27,0.13)
19  [0.22 - 0.44[: fran with credibility: 0.13 = min(0.13,0.33)
20  [0.78 - <[: frei with credibility: 0.53 = min(0.53,1.00)
21  [0.22 - 0.33[: gie with credibility: 0.13 = min(0.13,0.20)

```

```

22 [0.33 - 0.44[: goet with credibility: 0.07 = min(0.47,0.07)
23 [0.22 - 0.33[: ham with credibility: 0.07 = min(0.33,0.07)
24 [0.11 - 0.22[: han with credibility: 0.20 = min(0.33,0.20)
25 [0.78 - 0.89[: hei with credibility: 0.13 = min(0.13,0.27)
26 [0.56 - 0.67[: jena with credibility: 0.07 = min(0.13,0.07)
27 [0.33 - 0.44[: kiel with credibility: 0.20 = min(0.20,0.47)
28 [0.44 - 0.56[: koel with credibility: 0.07 = min(0.27,0.07)
29 [0.78 - <[: kons with credibility: 0.20 = min(0.20,1.00)
30 [0.56 - 0.89[: ksl with credibility: 0.13 = min(0.13,0.40)
31 [0.78 - 0.89[: leip with credibility: 0.07 = min(0.20,0.07)
32 [0.44 - 0.56[: main with credibility: 0.07 = min(0.07,0.13)
33 [0.56 - 0.67[: marb with credibility: 0.07 = min(0.07,0.07)
34 [0.56 - 0.89[: mnch with credibility: 0.20 = min(0.20,0.27)
35 [0.56 - 0.67[: mnst with credibility: 0.07 = min(0.20,0.07)
36 [0.78 - 0.89[: mu with credibility: 0.13 = min(0.13,0.47)
37 [0.56 - 0.67[: reg with credibility: 0.20 = min(0.20,0.27)
38 [0.56 - 0.78[: saar with credibility: 0.13 = min(0.13,0.20)
39 [0.78 - 0.89[: stu with credibility: 0.07 = min(0.13,0.07)
40 [0.44 - 0.56[: tri with credibility: 0.07 = min(0.13,0.07)
41 [0.67 - 0.78[: tueb with credibility: 0.13 = min(0.13,0.20)
42 [0.89 - <[: tum with credibility: 0.13 = min(0.13,1.00)
43 [0.56 - 0.67[: wrzb with credibility: 0.07 = min(0.20,0.07)

```

In the 9-tiles sorting result, shown in Listing 14.6, we notice for instance in Lines 3-4 that the RWTH Aachen is precisely rated into the 4th 9-tile ([0.33 – 0.44]), whereas the University Augsburg is less precisely rated conjointly into the 6th, the 7th and the 8th 9-tile ([0.56 – 0.89]). In Line 42, TU München appears best rated into the unique highest 9-tile ([0.89 – <]). All three rating results are supported by a $(0.07 + 1.0)/2 = 53.5\%$ majority of valuated subjects Footnote[31]. With the support of a 76.5% majority of valuated subjects (Line 20), the apparent most confident rating result is the one of University Freiburg (see also Fig. ?? and Fig. 14.4).

We shall now lexicographically sort these individual rating results per University, by *average* rated 9-tile limits and highest-rated upper 9-tile limit, into ordered, but not necessarily disjoint, enrolment quality quantiles.

```

1 >>> lqr.showHTMLQuantilesSorting(strategy='average')

```

With a special *graphviz* drawing of the `LearnedQuantilesRatingDigraph` instance `lqr`, we may, without requiring any specific ordering strategy, as well illustrate our 9-tiles rating-by-sorting result.

```

1 >>> lqr.exportRatingBySortingGraphViz(\n2 ...           'nineTilingDrawing', graphSize='12,12')\n3     *---- exporting a dot file for GraphViz tools --*\n4     Exporting to nineTilingDrawing.dot\n5     dot -Grankdir=TB -Tpng nineTilingDrawing.dot\\
6             -o nineTilingDrawing.png

```

In Fig. 14.7 we actually see the *skeleton* (transitive closure removed) of a partial order, where an oriented arc is drawn between Universities x and y when their 9-tiles sorting results are disjoint and the one of x is higher rated than the one of y. The rating for TU München (see Listing 14.6 Lines 45), for instance, is disjoint and higher rated than the one of the Universities Freiburg and Konstanz (Lines 23, 32). And, both the ratings of Feiburg and Konstanz are, however, not disjoint from the one, for instance, of the University of Stuttgart (Line 42).

The partial ranking, shown in Fig. 14.7, is in fact independent of any ordering strategy: - *average*, - *optimistic* - *pessimistic*, of overlapping 9-tiles sorting results, and confirms that the same Universities as with the previous rating-by-ranking approach, namely TU München, Freiburg, Konstanz, Stuttgart, Berlin Humboldt, Heidelberg and Leipzig appear top-rated. Similarly, the Universities of Duisburg, Bielefeld, Hanover, Bochum, Giessen, Düsseldorf and Hamburg give the lowest-rated group. The midfield here is again consisting of more or less the same Universities as the one observed in the previous rating-by-ranking approach (see Listing 14.3).

In the end, both the COPELAND rating-by-ranking, as well as the rating-by-sorting approach give luckily, in our case study here, very similar results. The first approach, with its *forced* linear ranking, determines on the one hand, precise enrolment quality equivalence classes; a result, depending potentially a lot on the actually applied ranking rule. The rating-by-sorting approach, on the other hand, only determines for each University a less precise but prudent rating of its individual enrolment quality, furthermore supported by a known majority of performance criteria significance; a somehow fairer and robust result, but, much less evident for easily comparing the apparent enrolment quality among Universities. Contradictorily, or sparsely valued Universities, for instance, will appear trivially rated into a large midfield of adjacent 9-tiles.

Let us conclude by saying that we prefer this latter rating-by-sorting approach; perhaps impreciser, due the case given, to missing and contradictory performance data; yet, well grounded in a powerful bipolar-valued logical and epistemic framework.

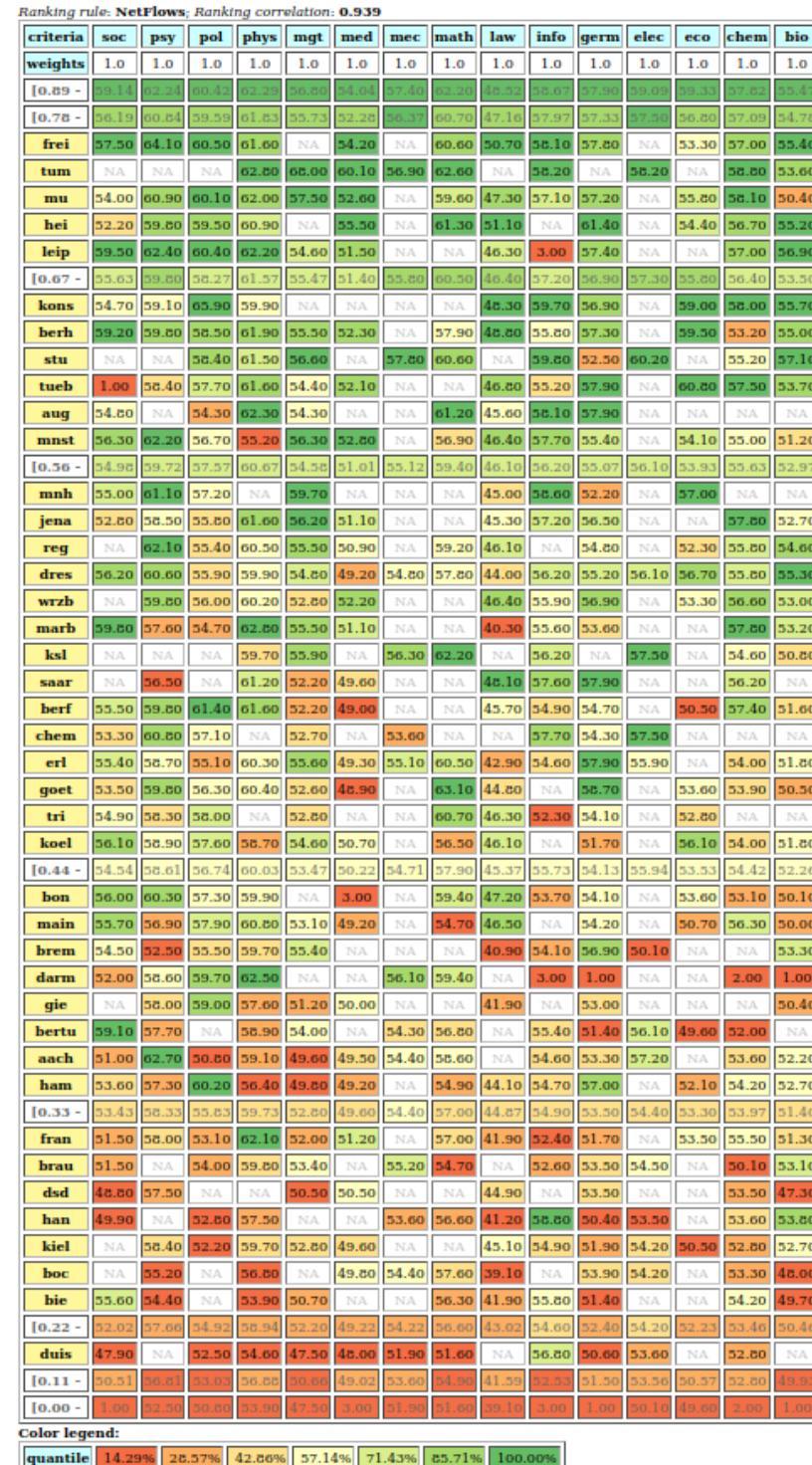


Fig. 14.4 Heatmap view of the 9-tiles rating-by-ranking result

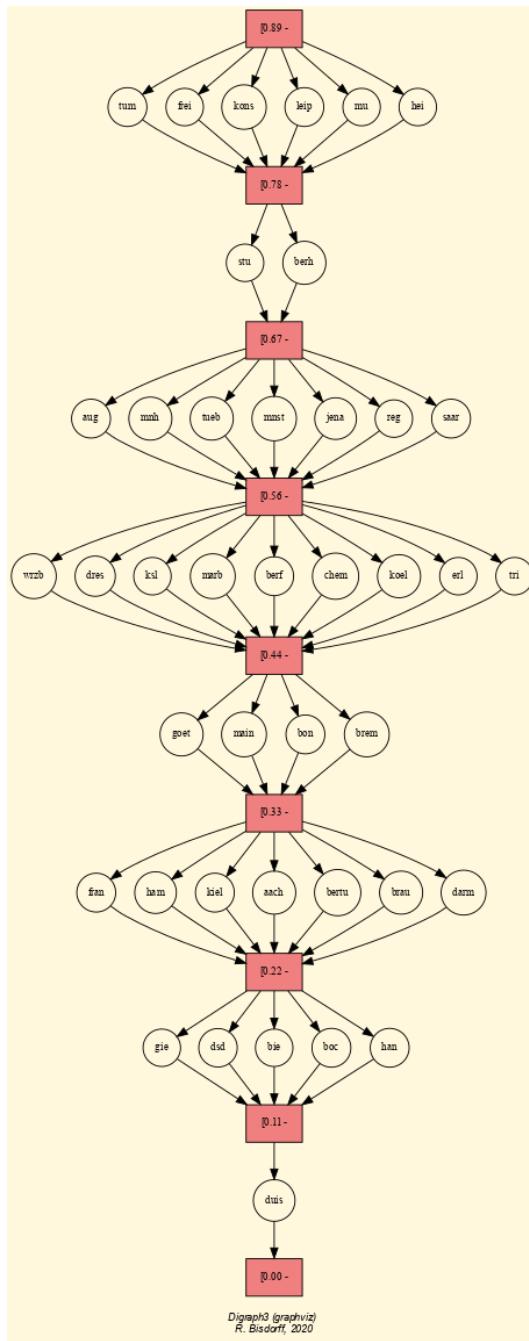


Fig. 14.5 Drawing of the 9-tiles rating-by-ranking result

Fig. 14.6 The ranked 9-tiles rating-by-sorting result. We may notice that the Universities: Augsburg, Kaiserslautern, Mannheim and Tübingen, for instance, show in fact the same average rated 9-tiles score of 0.725; yet, the rated upper 9-tile limit of Tuebingen reaches only 0.78, whereas the one of the other Universities reaches 0.89. Hence, Tuebingen is ranked below Augsburg, Kaiserslautern and Mannheim.

quantile limits	Ordering by average quantile class limits
[0.89-<	['tum']
[0.78-<	['frei', 'kons']
[0.78-0.89[['berh', 'hei', 'leip', 'mu', 'stu']
[0.56-0.89[['aug', 'ksl', 'mnch']
[0.67-0.78[['tueb']
[0.56-0.78[['saar']
[0.56-0.67[['dres', 'jena', 'marb', 'mnst', 'reg', 'wrzb']
[0.44-0.67[['berf']
[0.44-0.56[['bon', 'chem', 'erl', 'koel', 'main', 'tri']
[0.22-0.56[['darm']
[0.33-0.44[['aach', 'brau', 'brem', 'goet', 'kiel']
[0.22-0.44[['bertu', 'fran']
[0.22-0.33[['boc', 'dsd', 'gie', 'ham']
[0.11-0.22[['bie', 'han']
[0.00-0.11[['duis']

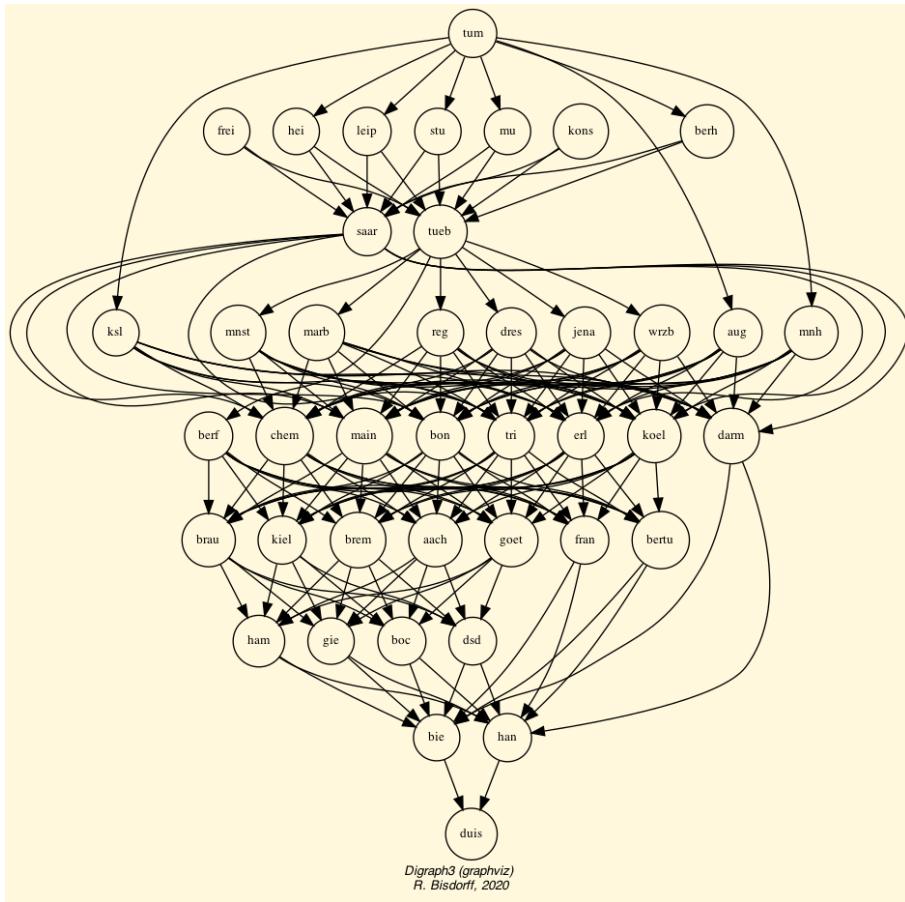


Fig. 14.7 Graphviz drawing of the 9-tiles sorting digraph.

Chapter 15

Exercises

Abstract We propose hereafter some decision problems which may serve as exercises and exam questions in an *Algorithmic Decision Theory* Course. They cover *selection*, *ranking* and *rating* decision problems.

The exercises we propose in this Chapter are marked as follows: § (*warming up*), §§ (*home work*), §§§ (*research work*). Solutions should be supported both by computational Python code using the DIGRAPH3 3 programming resources as well as by methodological and algorithmic arguments from the *Algorithmic Decision Theory* Lectures [ADT-2020].

15.1 Who will receive the best student award? (§)

15.1.1 Data

Below in Table 15.1 you see the actual grades obtained by four students : *Ariana* (A), *Bruce* (B), *Clare* (C) and *Daniel* (D) in five courses: C1, C2, C3, C4 and C5 weighted by their respective ECTS points.

Table 15.1 Grades obtained by the students

Course ECTS	C1	C2	C3	C4	C5
	2	3	4	2	4
Ariana (A)	11	13	9	15	11
Bruce (B)	12	9	13	10	13
Clare (C)	8	11	14	12	14
Daniel (D)	15	10	12	8	13

The grades shown in Table 15.1 are given on an ordinal performance scale from 0 pts (weakest) to 20 pts (highest). Assume that the grading admits a preference threshold of 1 points. 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 *best* amongst these four students.

15.1.2 Questions

1. Edit a :ref:`PerformanceTableau` instance with the data shown above.
2. Who would you nominate?
3. Explain and motivate your selection algorithm.
4. Assume that the grading may actually admit an indifference threshold of 1 point and a preference threshold of 2 points. How stable is your result with respect to the actual preference discrimination power of the grading scale?

15.2 How to fairly rank movies (§)

15.2.1 Data

- File `graffiti03.py` contains a performance tableau about the rating of movies to be seen in the city of Luxembourg, February 2003. Its content is shown in Fig. 15.1 below.

```

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)

```

The critic's opinions are expressed on a 7-graded scale: -2 (two zeros, *I hate*), -1 (one zero, *I don't like*), 1 (one star, *maybe*), 2 (two stars, *good*), 3 (three stars, *excellent*), 4 (four stars, *not to be missed*), and 5 (five stars, *a master piece*). Notice the many missing data (NA) when a critic had not seen the respective movie. Mind also that the ratings of two movie critics ('jh' and 'vt') are given a higher significance weight.

criteria	germ	pol	psy	soc	law	eco	mgt	bio	med	phys	chem	math	info	elec	mec
weights	+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	+1.00	+1.00
aach	53.3	50.8	62.7	51.0	NA	NA	49.6	52.2	49.5	59.1	53.6	58.6	54.6	57.2	54.4
aug	57.9	54.3	NA	54.8	45.6	NA	54.3	NA	NA	62.3	NA	61.2	58.1	NA	NA
berf	54.7	61.4	59.8	55.5	45.7	50.5	52.2	51.6	49.0	61.6	57.4	NA	54.9	NA	NA
berh	57.3	58.5	59.8	59.2	48.8	59.5	55.5	55.0	52.3	61.9	53.2	57.9	55.8	NA	NA
bertu	51.4	NA	57.7	59.1	NA	49.6	54.0	NA	NA	58.9	52.0	56.8	55.4	56.1	54.3
bie	51.4	NA	54.4	55.6	41.9	NA	50.7	49.7	NA	53.9	54.2	56.3	55.8	NA	NA
boc	53.9	NA	55.2	NA	39.1	NA	NA	48.0	49.8	56.8	53.3	57.6	NA	54.2	54.4
bon	54.1	57.3	60.3	56.0	47.2	53.6	NA	50.1	3.0	59.9	53.1	59.4	53.7	NA	NA
brau	53.5	54.0	NA	51.5	NA	NA	53.4	53.1	NA	59.8	50.1	54.7	52.6	54.5	55.2
brem	56.9	55.5	52.5	54.5	40.9	NA	55.4	53.3	NA	59.7	NA	NA	54.1	50.1	NA
chem	54.3	57.1	60.8	53.3	NA	NA	52.7	NA	NA	NA	NA	NA	57.7	57.5	53.6
darm	1.0	59.7	58.6	52.0	NA	NA	NA	1.0	NA	62.5	2.0	59.4	3.0	NA	56.1
dres	55.2	55.9	60.6	56.2	44.0	56.7	54.8	55.3	49.2	59.9	55.8	57.8	56.2	56.1	54.8
dsd	53.5	NA	57.5	48.8	44.9	NA	50.5	47.3	50.5	NA	53.5	NA	NA	NA	NA
duis	50.6	52.5	NA	47.9	NA	NA	47.5	NA	48.0	54.6	52.8	51.6	56.8	53.6	51.9
erl	57.9	55.1	58.7	55.4	42.9	NA	55.6	51.8	49.3	60.3	54.0	60.5	54.6	55.9	55.1
fran	51.7	53.1	58.0	51.5	41.9	53.5	52.0	51.3	51.2	62.1	55.5	57.0	52.4	NA	NA
frei	57.8	60.5	64.1	57.5	50.7	53.3	NA	55.4	54.2	61.6	57.0	60.6	58.1	NA	NA
gie	53.0	59.0	58.0	NA	41.9	NA	51.2	50.4	50.0	57.6	NA	NA	NA	NA	NA
goet	58.7	56.3	59.8	53.5	44.8	53.6	52.6	50.5	48.9	60.4	53.9	63.1	NA	NA	NA
ham	57.0	60.2	57.3	53.6	44.1	52.1	49.8	52.7	49.2	56.4	54.2	54.9	54.7	NA	NA
han	50.4	52.8	NA	49.9	41.2	NA	NA	53.8	NA	57.5	53.6	56.6	58.8	53.5	53.6
hei	61.4	59.5	59.8	52.2	51.1	54.4	NA	55.2	55.5	60.9	56.7	61.3	NA	NA	NA
jena	56.5	55.8	58.5	52.8	45.3	NA	56.2	52.7	51.1	61.6	57.8	NA	57.2	NA	NA
kiel	51.9	52.2	58.4	NA	45.1	50.5	52.8	52.7	49.6	59.7	52.8	NA	54.9	54.2	NA
koel	51.7	57.6	58.9	56.1	46.1	56.1	54.6	51.8	50.7	58.7	54.0	56.5	NA	NA	NA
kons	56.9	65.9	59.1	54.7	48.3	59.0	NA	55.7	NA	59.9	58.0	NA	59.7	NA	NA
ksl	NA	55.9	50.8	NA	59.7	54.6	62.2	56.2	57.5						
leip	57.4	60.4	62.4	59.5	46.3	NA	54.6	56.9	51.5	62.2	57.0	NA	3.0	NA	NA
main	54.2	57.9	56.9	55.7	46.5	50.7	53.1	50.0	49.2	60.8	56.3	54.7	NA	NA	NA
marb	53.6	54.7	57.6	59.8	40.3	NA	55.5	53.2	51.1	62.8	57.8	NA	55.6	NA	NA
mnh	52.2	57.2	61.1	55.0	45.0	57.0	59.7	NA	NA	NA	NA	NA	58.6	NA	NA
mnst	55.4	56.7	62.2	56.3	46.4	54.1	56.3	51.2	52.8	55.2	55.0	56.9	57.7	NA	NA
mu	57.2	60.1	60.9	54.0	47.3	55.8	57.5	50.4	52.6	62.0	58.1	59.6	57.1	NA	NA
reg	54.8	55.4	62.1	NA	46.1	52.3	55.5	54.6	50.9	60.5	55.8	59.2	NA	NA	NA
saar	57.9	NA	56.5	NA	48.1	NA	52.2	NA	49.6	61.2	56.2	NA	57.6	NA	NA
stu	52.5	58.4	NA	NA	NA	NA	56.6	57.1	NA	61.5	55.2	60.6	59.8	60.2	57.8
tri	54.1	58.0	58.3	54.9	46.3	52.8	52.8	NA	NA	NA	NA	60.7	52.3	NA	NA
tueb	57.9	57.7	58.4	1.0	46.8	60.8	54.4	53.7	52.1	61.6	57.5	NA	55.2	NA	NA
tum	NA	68.0	53.6	60.1	62.8	58.8	62.6	58.2	56.9						
wrbz	56.9	56.0	59.8	NA	46.4	53.3	52.8	53.0	52.2	60.2	56.6	NA	55.9	NA	NA

Color legend:

quantile	14.29%	28.57%	42.86%	57.14%	71.43%	85.71%	100.00%
----------	--------	--------	--------	--------	--------	--------	---------

Fig. 15.1 Graffiti magazine's movie ratings from February 2003

15.2.2 Questions

1. The Graffiti magazine suggest a best rated movie with the help of an average number of stars, ignoring the missing data and any significance weights of the critics. By taking into account missing data and varying significance weights, how may one find the best rated movie without computing any average rating scores?

2. How would one rank these movies so as to at best respect the weighted rating opinions of each movie critic?
3. In what ranking position would appear a movie not seen by any movie critic? Confirm computationally the answer by adding such a fictive, *not at all evaluated*, movie to the given performance tableau instance.
4. How robust are the preceding results when the significance weights of the movie critics are considered to be only ordinal grades ?

15.3 What is your best choice recommendation? (§)

15.3.1 Data Footnote[43]

A person, who wants to buy a TV set, retains after a first selection, eight potential TV models. To make up her choice these eight models were evaluated with respect to three decision objectives of equal importance: - *Costs* of the set (to be minimized); - *Picture and Sound* quality of the TV (to be maximized); - *Maintenace contract* quality of the provider (to be maximized).

The *Costs* objective is assessed by the price of the TV set (criterion *Pr* to be minimized). *Picture* quality (criterion *Pq*), *Sound* quality (criterion *Sq*) and *Maintenace 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*).

The actual evaluation data are gathered in Table 15.2 below.

Table 15.2 Performance evaluations of the potential TV sets

Criteria Significance	Pr (€) 2	Pq 1	Sq 1	Mq
Model T1	-1300	2	2	0
Model T2	-1200	2	2	1
Model T3	-1150	2	1	1
Model T4	-1000	1	1	-1
Model T5	-950	1	1	0
Model T6	-950	0	1	-1
Model T7	-900	1	0	-1
Model T8	-900	0	0	0

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.

15.3.2 Questions

1. Edit a new `PerformanceTableau` instance with the data shown above and illustrate its content by best showing objectives, criteria, decision alternatives and performance table. If needed, write adequate python code.
2. What is the best TV set to recommend?
3. Illustrate your best choice recommendation with an adequate `graphviz` drawing.
4. Explain and motivate your selection algorithm.
5. Assume that the qualitative criteria: *Picture quality* ('Pq'), *Sound quality* ('Sq'), and *Maintenace 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 changing these criteria significance weights?

15.4 What is the best public policy? (§§)

15.4.1 Data files

- File `perfTab_1.py` [Footnote] contains a 3 Objectives performance tableau with 100 performance records concerning public policies evaluated with respect to an *economic*, a *societal* and an *environmental* public decision objective.
- File `historicalData_1.py` [footnote] contains a performance tableau of the same kind with 2000 historical performance records.

15.4.2 Questions

1. Illustrate the content of the given file `perfTab_1.py` performance tableau by best showing *objectives*, *criteria*, *decision alternatives* and *performance table*. If needed, write adequate python code.
2. Construct the corresponding bipolar-valued outranking digraph. How *confident* and/or *robust* are the apparent outranking situations?
3. 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.
4. How would you rate your 100 public policies into relative deciles classes?
5. Using the given historical records in file `historicalData_1.py`, how would you rate your 100 public policies into absolute deciles classes? Explain

the differences you may observe between the absolute and the previous relative rating result.

6. Select among your 100 potential policies a shortlist of up to 15 potential best policies, all reaching an absolute performance quantile of at least 66.67%.
7. Based on the previous best policies shortlist (see Question 6), what is your eventual best-choice recommendation? Is it perhaps an unopposed best choice by all three objectives?

15.5 A fair diploma validation decision (\$\$\$)

15.5.1 Data

Use the RandomAcademicPerformanceTableau constructor from the DIGRAPH3 Python resources for generating realistic random students performance tableaux concerning a curriculum of nine ECTS weighted Courses. Assume that all the gradings are done on an integer scale from 0 (weakest) to 20 (best). It is known that all grading procedures are inevitably imprecise; therefore we will assume an indifference threshold of 1 point and a preference 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.

15.5.2 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.

Part IV

Advanced topics

“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)

Chapter 16

Ordinal correlation equals bipolar-valued relational equivalence

Abstract

16.1 Coping with missing data

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 like elections, 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.

Let us consider such a performance tableau in file `graffiti07.py` Footnote[] gathering a Movie Magazine's rating of some movies that could actually be seen in town Footnote[1] (see Fig. 16.1).

```
1 >>> from outrankingDigraphs import \
2 ...     PerformanceTableau
3 >>> t = PerformanceTableau('graffiti07')
4 >>> t.showHTMLPerformanceTableau(\
5 ...             title='Graffiti Star wars', \
6 ...             ndigits=0)
```

15 journalists and movie critics provide here their rating of 25 movies: 5 stars (*masterpiece*), 4 stars (*must be seen*), 3 stars (*excellent*), 2 stars (*good*), 1 star (*could be seen*), -1 star (*I do not like*), -2 (*I hate*), NA (*not seen*).

To aggregate all the critics' rating opinions, the *Graffiti* magazine provides for each movie a global score computed as an average grade, just ignoring the *not seen* data. These averages are thus not computed on comparable denominators; some

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	3.00	1.00	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 Graffiti magazine's movie ratings from September 2007.

critics do indeed use a more or less extended range of grades. The movies not seen by critic *SJ*, for instance, are favored, as this critic is more severe than others in her grading. Dropping the movies that were not seen by all the critics is here not possible either, as no one of the 25 movies was actually seen by all the critics. Providing any value for the missing data will as well always somehow falsify any global value scoring. What to do ?

A better approach is to rank the movies on the basis of pairwise bipolar-valued “*at least as well rated as*” opinions. Under this epistemic argumentation approach, missing data are naturally treated as opinion abstentions and hence do not falsify the logical computations. Such a ranking of the 25 movies is provided, for instance, by the **heatmap** view shown in Fig. 16.2.

```

1 >>> t.showHTMLPerformanceHeatmap(\n2 ...                                Correlations=True,\n3 ...                                rankingRule='NetFlows',\n4 ...                                ndigits=0)

```

There is no doubt that movie 'mv_QS', with 6 *must be seen* marks, is correctly best-ranked and the movie 'mv_TV' is worst-ranked with five *don't like* marks.

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	1	2	3	NA	1	-1	NA	4	NA	1	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 Graffiti magazine's ordered movie ratings from September 2007

16.2 Modelling pairwise bipolar-valued rating opinions

Let us explicitly construct the underlying bipolar-valued outranking digraph and consult in Fig. 16.3 the pairwise characteristic values we observe between the two best-ranked movies, namely 'mv_QS' and 'mv_RR'.

```

1 >>> from outrankingDigraphs import \
2 ...     BipolarOutrankingDigraph
3 >>> g = BipolarOutrankingDigraph(t)
4 >>> g.recodeValuation(-19,19) # integer characteristics
5 >>> g.showHTMLPairwiseOutrankings('mv_QS','mv_RR')

```

Six out of the fifteen critics have not seen one or the other of these two movies. Notice the higher significance (3) that is granted to two locally renowned movie critics, namely 'JH' and 'VT'. Their opinion counts for three times the opinion of the other critics. All nine critics that have seen both movies, except critic 'MR', state that 'mv_QS' is rated at least as well as 'mv_RR' and the balance of positive against negative opinions amounts to +11, a characteristic value which positively validates the outranking situation with a majority of $(11/19 + 1.0)/2.0 = 79\%$.

Pairwise Comparison										
Comparing actions : (mv_QS,mv_RR)										
crit.	wght.	g(x)	g(y)	diff	ind	wp	p	concord	wv	polarisation
AP	1.00	NA	3.00							0.00
AS	1.00	3.00	2.00	+1.00	None	None	None	+1.00		
CF	1.00	2.00	NA						0.00	
CS	1.00	NA	1.00						0.00	
DR	1.00	4.00	4.00	+0.00	None	None	None	+1.00		
FG	1.00	3.00	NA						0.00	
GS	1.00	NA	4.00						0.00	
JH	3.00	4.00	3.00	+1.00	None	None	None	+3.00		
JPT	1.00	4.00	3.00	+1.00	None	None	None	+1.00		
MR	1.00	3.00	4.00	-1.00	None	None	None	-1.00		
RR	1.00	3.00	3.00	+0.00	None	None	None	+1.00		
SF	1.00	4.00	3.00	+1.00	None	None	None	+1.00		
SJ	1.00	NA	NA						0.00	
TD	1.00	4.00	4.00	+0.00	None	None	None	+1.00		
VT	3.00	4.00	2.00	+2.00	None	None	None	+3.00		

Valuation in range: -19.00 to +19.00; global concordance: +11.00

Pairwise Comparison										
Comparing actions : (mv_RR,mv_QS)										
crit.	wght.	g(x)	g(y)	diff	ind	wp	p	concord	wv	polarisation
AP	1.00	3.00	NA						0.00	
AS	1.00	2.00	3.00	-1.00	None	None	None	-1.00		
CF	1.00	NA	2.00					0.00		
CS	1.00	1.00	NA					0.00		
DR	1.00	4.00	4.00	+0.00	None	None	None	+1.00		
FG	1.00	NA	3.00					0.00		
GS	1.00	4.00	NA					0.00		
JH	3.00	3.00	4.00	-1.00	None	None	None	-3.00		
JPT	1.00	3.00	4.00	-1.00	None	None	None	-1.00		
MR	1.00	4.00	3.00	+1.00	None	None	None	+1.00		
RR	1.00	3.00	3.00	+0.00	None	None	None	+1.00		
SF	1.00	3.00	4.00	-1.00	None	None	None	-1.00		
SJ	1.00	NA	NA					0.00		
TD	1.00	4.00	4.00	+0.00	None	None	None	+1.00		
VT	3.00	2.00	4.00	+2.00	None	None	None	-3.00		

Valuation in range: -19.00 to +19.00; global concordance: -5.00

Fig. 16.3 Pairwise comparison of the two best-ranked movies.

The complete table of pairwise majority margins of global “*at least as well rated as*” opinions, ranked by the same rule as shown in the heatmap in Fig. 16.3, may be shown in Fig. 16.4.

```

1 >>> ranking = g.computeNetFlowsRanking()
2 >>> g.showHTMLRelationTable(actionsList=ranking, ndigits=0, \
3 ...     tableTitle='Bipolar characteristic values of\' \
4 ...     "rated at least as good as" situations')

```

Bipolar characteristic values of “*rated at least as good as*” situations

rtx	sty	mv_QS	mv_RR	mv_NP	mv_HS	mv_HS	mv_SM	mv_JB	mv_PF	mv_FC	mv_TP	mv_CM	mv_DF	mv_TM	mv_DJ	mv_AL	mv_RG	mv_MB	mv_GF	mv_HP	mv_BI	mv_DF	mv_FF	mv_GG	mv_TF
mv_QS	-	11	12	11	10	9	14	9	11	13	9	10	9	9	7	6	9	9	7	6	5	9	15	8	8
mv_RR	-5	-	5	7	8	6	4	5	2	9	10	9	12	9	10	8	9	6	10	13	10	9			
mv_DG	-8	9	-	9	10	5	9	7	8	13	7	7	10	10	6	8	10	7	9	5	6	7	9	9	
mv_NP	-7	5	7	-	10	3	7	9	8	11	9	8	12	8	7	6	4	6	8	6	5	7	12	10	
mv_HN	-10	4	8	8	-	5	9	9	8	10	10	7	10	6	8	10	7	6	8	8	5	7	13	9	
mv_HS	-3	2	5	5	3	-	2	5	6	9	5	10	7	0	1	10	9	8	9	7	11	14	9	11	
mv_SM	-8	6	7	5	7	6	-	6	6	10	12	9	10	6	4	9	9	10	7	9	7	11	15	11	
mv_JB	-9	-1	5	1	3	4	8	-	9	6	6	13	6	8	9	9	9	8	8	6	5	11	15	12	
mv_PF	-7	2	6	0	4	3	6	11	-	5	4	10	5	7	6	8	8	8	5	6	5	9	13	9	
mv_FC	-9	5	11	9	8	2	8	8	7	-	10	6	11	5	3	8	6	9	9	6	7	10	12	10	
mv_TP	-7	4	-1	3	2	3	0	-	4	-4	0	-	6	11	6	4	5	7	6	7	7	9	14	10	
mv_CM	-8	-1	-5	-4	-3	-3	-1	5	-2	-2	4	-	3	8	8	9	10	7	5	7	3	9	14	11	
mv_DF	-9	-2	-2	-2	-2	-2	-1	0	0	1	-1	5	1	-	4	6	-1	6	8	6	5	6	8	13	
mv_TM	-3	-7	-6	-2	-1	-1	-1	-1	-2	-2	-1	-2	-2	-2	-2	-2	-2	-2	7	6	9	7	5	6	
mv_DJ	-7	-8	-4	-3	2	4	2	1	-2	-1	0	0	0	4	-	3	5	4	3	6	4	4	3	1	
mv_AL	-4	0	-4	-6	-6	1	-1	-3	-4	-2	3	1	3	2	3	-	2	2	5	1	3	1	5	3	
mv_RG	-7	-7	-6	-4	-1	-2	-1	-5	-4	-2	-3	-2	-2	1	3	4	-	1	0	6	4	8	9	4	
mv_MB	-9	-8	-5	-4	-4	-9	-4	-1	-2	-2	-1	-2	-2	2	3	-	2	4	4	4	2	6	6	8	
mv_GF	-5	-8	-7	-6	-8	-4	-3	0	1	-5	-5	1	-2	-1	-3	-3	0	6	-	5	2	5	3	3	
mv_HP	-4	-5	-5	-2	-4	-3	-4	-4	-3	-1	-3	1	-7	-2	-1	2	-2	-1	-	3	6	5	7	8	
mv_BI	-5	-4	-6	-1	-1	-7	-5	-5	-5	-3	-1	-3	0	-3	-2	1	-2	0	2	1	-	1	-1	4	
mv_DF	-9	-6	-5	-5	-3	-3	-3	-7	-6	-1	-7	-0	-5	0	1	4	0	1	8	5	-	6	4	8	
mv_FF	-11	-11	-7	-10	-5	0	-3	-9	-9	-8	-2	-6	1	-1	5	-1	9	0	1	7	5	12	-	9	
mv_GG	-6	-8	-7	-4	-5	-7	-9	-10	-9	-4	-2	-7	3	-3	1	-2	-2	-1	-3	5	-2	3	-	2	
mv_TF	-8	-7	-7	-5	-7	-11	-9	-8	-8	-7	-6	-8	-3	-6	-3	-1	-1	-4	-3	0	2	2	-1	2	

Valuation domain: [-19.00; +19.00]

Fig. 16.4 Pairwise majority margins of “*at least as well rated as*” rating opinions

Positive characteristic values, validating a global “*at least as well rated as*” opinion are marked in light green (see Fig. 16.4). Whereas negative characteristic values, invalidating such a global opinion, are marked in light red. We may by the way notice that the best-ranked movie ’mv_QS’ is indeed a CONDORCET winner, i.e. better rated than all the other movies by a 65% majority of critics. This majority may be

assessed from the average determinateness of the given bipolar-valued outranking digraph g .

```
1 >>> print( '%.0f%%' % g.computeDeterminateness(InPercents=True) )
2 65%
```

Notice also the indeterminate situation we observe, for instance, when comparing movie 'mv_PE' with movie 'mv_NP'.

```
1 >>> g.showHTMLPairwiseComparison('mv_PE','mv_NP')
```

Pairwise Comparison

Comparing actions : (mv_PE,mv_NP)

crit.	wght.	g(x)	g(y)	diff	ind	wp	p	concord	wv	v	polarisation
AP	1.00	3.00	NA						0.00		
AS	1.00	4.00	1.00	+3.00	None	None	None	+1.00			
CF	1.00	2.00	3.00	-1.00	None	None	None	-1.00			
CS	1.00	NA	1.00						0.00		
DR	1.00	3.00	NA						0.00		
FG	1.00	1.00	3.00	-2.00	None	None	None	-1.00			
GS	1.00	NA	2.00						0.00		
JH	3.00	2.00	3.00	-1.00	None	None	None	-3.00			
JPT	1.00	4.00	3.00	+1.00	None	None	None	+1.00			
MR	1.00	2.00	3.00	-1.00	None	None	None	-1.00			
RR	1.00	NA	2.00						0.00		
SF	1.00	3.00	3.00	+0.00	None	None	None	+1.00			
SJ	1.00	3.00	NA						0.00		
TD	1.00	NA	NA						0.00		
VT	3.00	3.00	3.00	+0.00	None	None	None	+3.00			

Valuation in range: -19.00 to +19.00; global concordance: +0.00

Fig. 16.5 Indeterminate pairwise comparison example.

Only eight, out of the fifteen critics, have seen both movies and the positive opinions do neatly balance the negative ones. A global statement that 'mv_PE' is "at least as well rated as" 'mv_NP' may in this case hence neither be validated, nor invalidated; a preferential situation that cannot be modelled with any scoring approach.

It is fair, however, to eventually mention here that the *Graffiti* magazine's average scoring method is actually showing a very similar ranking. Indeed, average scores usually confirm well all evident pairwise comparisons, yet *enforce* comparability for all less evident ones.

Notice finally the ordinal correlation τ_{au} values in Fig. 16.2 3rd row. Computing these ordinal correlation indexes is the subject of the next Section.

16.3 KENDALL's tau index

M. G. Kendall ([KEN-1938p]) 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. 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).

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, KENDALL's original *tau* definition implemented in fact the bipolar-valued negation of the non equivalence of two linear orders:

$$\tau = 1 - 2 \frac{In}{n(n-1)} = -\left(2 \frac{In}{n(n-1)} - 1\right) = 2 \frac{Co}{n(n-1)} - 1, \quad (16.1)$$

i.e. the normalized majority margin of equivalently oriented irreflexive pairs.

Let R_1 and R_2 be two random crisp relations defined on a same set of 5 alternatives. We may compute KENDALL's *tau* index as follows.

Listing 16.1 Computing a relational equivalence digraph

```

1 >>> from randomDigraphs import RandomDigraph
2 >>> R1 = RandomDigraph(order=5,Bipolar=True)
3 >>> R2 = RandomDigraph(order=5,Bipolar=True)
4 >>> from digraphs import EquivalenceDigraph
5 >>> E = EquivalenceDigraph(R1,R2)
6 >>> E.showRelationTable(ReflexiveTerms=False)
7     * ---- Relation Table ----
8     r(<=>) | 'a1'   'a2'   'a3'   'a4'   'a5'
9     -----|-----
10    'a1' |   -   -1.00   1.00   -1.00   1.00
11    'a2' |   -1.00   -   -1.00   1.00   -1.00
12    'a3' |   -1.00   -1.00   -   1.00   1.00
13    'a4' |   -1.00   1.00   -1.00   -   1.00
14    'a5' |   -1.00   1.00   -1.00   1.00   -
15    Valuation domain: [-1.00;1.00]
16 >>> E.correlation
17     {'correlation': -0.1, 'determination': 1.0}

```

In the table of the equivalence relation ($R_1 \Leftrightarrow R_2$) above (see Listing 16.1 Lines 10-14), we observe that the normalized 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 determined and even partially indeterminate relations? May we proceed in a similar way?

16.4 Bipolar-valued relational equivalence

Let us now consider two randomly bipolar-valued digraphs R_1 and R_2 of order five.

Listing 16.2 Two random bipolar-valued digraphs

```

1 >>> R1 = RandomValuationDigraph(order=5, seed=1)
2 >>> R1.showRelationTable(ReflexiveTerms=False)
3   * ---- Relation Table ----
4   r(R1) |  'a1'  'a2'  'a3'  'a4'  'a5'
5   -----|-----
6   'a1' |    -   -0.66  0.44   0.94  -0.84
7   'a2' |  -0.36   -   -0.70   0.26   0.94
8   'a3' |  0.14   0.20   -     0.66  -0.04
9   'a4' |  -0.48  -0.76  0.24   -     -0.94
10  'a5' |  -0.02   0.10  0.54   0.94   -
11  Valuation domain: [-1.00;1.00]
12 >>> R2 = RandomValuationDigraph(order=5, seed=2)
13 >>> R2.showRelationTable(ReflexiveTerms=False)
14   * ---- Relation Table ----
15  r(R2) |  'a1'  'a2'  'a3'  'a4'  'a5'
16  -----|-----
17  'a1' |    -   -0.86  -0.78  -0.80  -0.08
18  'a2' |  -0.58   -   0.88   0.70  -0.22
19  'a3' |  -0.36   0.54   -     -0.46  0.54
20  'a4' |  -0.92   0.48   0.74   -     -0.60
21  'a5' |   0.10   0.62   0.00   0.84   -
22  Valuation domain: [-1.00;1.00]
```

We may notice in the relation tables shown above that 9 pairs, like (a1,a2) or (a3,a2) for instance, appear equivalently oriented (see Listing 16.2 Lines 6,17 or 8,19). The EquivalenceDigraph class implements this relational equivalence relation between digraphs R_1 and R_2 (see Listing 16.3).

Listing 16.3 Bipolar-valued Equivalence Digraph

```

1 >>> eq = EquivalenceDigraph(R1, R2)
2 >>> eq.showRelationTable(ReflexiveTerms=False)
3   * ---- Relation Table ----
4   r(<=>) |  'a1'  'a2'  'a3'  'a4'  'a5'
5   -----|-----
6   'a1' |    -   0.66  -0.44  -0.80  0.08
7   'a2' |  0.36   -   -0.70   0.26  -0.22
8   'a3' | -0.14   0.20   -     -0.46  -0.04
9   'a4' |  0.48  -0.48   0.24   -     0.60
10  'a5' | -0.02   0.10   0.00   0.84   -
11  Valuation domain: [-1.00;1.00]
```

In our bipolar-valued epistemic logic, logical disjunctions and conjunctions are implemented as max, respectively min operators. Notice also that the logical equivalence ($R_1 \Leftrightarrow R_2$) corresponds to a double implication ($R_1 \Rightarrow R_2 \wedge R_2 \Rightarrow R_1$) and that the implication ($R_1 \Rightarrow R_2$) is logically equivalent to the disjunction ($\neg R_1 \vee R_2$).

When $r(xR_1y)$ and $r(xR_2y)$ denote the bipolar-valued characteristic values of relation R_1 , resp. R_2 , we may hence compute as follows a majority margin $M(R_1 \Leftrightarrow$

R_2) between equivalently and not equivalently oriented irreflexive pairs (x,y) .

$$M(R_1 \Leftrightarrow R_2) = \sum_{(x \neq y)} \left[\min \left(\max (-r(xR_1y), r(xR_2y)), \max (-r(xR_2y), r(xR_1y)) \right) \right]. \quad (16.2)$$

$M(R_1 \Leftrightarrow R_2)$ is thus given by the sum of the non reflexive terms of the relation table of eq , the relation equivalence digraph computed above (see Listing 16.3).

In the crisp case, $M(R_1 \Leftrightarrow R_2)$ is now normalized with the maximum number of possible irreflexive pairs, namely $n(n - 1)$. In a generalized r -valued case, the maximal possible equivalence majority margin M corresponds to the sum D of the conjoint determinations of (xR_1y) and (xR_2y) (see [BIS-2012pl]).

$$D = \sum_{x \neq y} \min \left[\text{abs}(r(xR_1y)), \text{abs}(r(xR_2y)) \right]. \quad (16.3)$$

Thus, we obtain in the general r -valued case:

$$\tau(R_1, R_2) = \frac{M(R_1 \Leftrightarrow R_2)}{D}. \quad (16.4)$$

$\tau(R_1, R_2)$ corresponds thus to a classical ordinal correlation index, but restricted to the conjointly determined parts of the given relations R_1 and R_2 . In the limit case of two crisp linear orders, D equals $n(n - 1)$, i.e. the number of irreflexive pairs, and we recover hence KENDALL's original *tau* index definition.

It is worthwhile noticing that the ordinal correlation index $\tau(R_1, R_2)$ we obtain above corresponds to the ratio of

- $r(R_1 \Leftrightarrow R_2) = \frac{M(R_1 \Leftrightarrow R_2)}{n(n-1)}$: the normalized majority margin of the pairwise *relational* equivalence statements, also called *valued ordinal correlation*, and
- $d = \frac{D}{n(n-1)}$: the normalized determination of the corresponding pairwise relational equivalence statements, in fact the *determinateness* of the relational equivalence digraph.

We have thus successfully *out-factored* the epistemic determination effect from the ordinal correlation effect. With completely determined relations, $\tau(R_1, R_2) = r(R_1 \Leftrightarrow R_2)$. By convention, we set the ordinal correlation with a completely indeterminate relation, i.e. when $D = 0$, to the indeterminate correlation value 0.0. With uniformly chosen random r -valued relations, the expected τ index is 0.0, denoting in fact an indeterminate correlation. The corresponding expected normalized determination d is about 0.333 (see [BIS-2012pl]).

We may verify these relations with help of the corresponding equivalence digraph eq (see Listing 16.4).

Listing 16.4 Computing the ordinal correlation index from the equivalence digraph

```
1 >>> eq = EquivalenceDigraph(R1, R2)
2 >>> M = Decimal('0'); D = Decimal('0')
```

```

3 >>> n2 = eq.order*(eq.order - 1)
4 >>> for x in eq.actions:
5 ...     for y in eq.actions:
6 ...         M += eq.relation[x][y]
7 ...         D += abs(eq.relation[x][y])
8 >>> print('r(R1<=>R2) = %+.3f, d = %.3f, tau = %+.3f' %
9 ...           (M/n2,D/n2,M/D))
9   r(R1<=>R2) = +0.026, d = 0.356, tau = +0.073

```

In general we simply use the `computeOrdinalCorrelation()` method which renders a dictionary with a correlation (τ) and a determination (d) attribute. We may recover $r(\Leftrightarrow)$ by multiplying τ with d (see Listing 16.5 Line 4).

Listing 16.5 Directly computing the ordinal correlation index

```

1 >>> corrR1R2 = R1.computeOrdinalCorrelation(R2)
2 >>> tau = corrR1R2['correlation']
3 >>> d = corrR1R2['determination']
4 >>> r = tau * d
5 >>> print('tau(R1,R2) = %+.3f, d = %.3f,\ \
6 ...     r(R1<=>R2) = %+.3f' % (tau, d, r))
7   tau(R1,R2) = +0.073, d = 0.356, r(R1<=>R2) = +0.026

```

We provide for convenience a direct `showCorrelation` method:

```

1 >>> corrR1R2 = R1.computeOrdinalCorrelation(R2)
2 >>> R1.showCorrelation(corrR1R2)
3   Correlation indexes:
4     Extended Kendall tau      : +0.073
5     Epistemic determination   : 0.356
6     Bipolar-valued equivalence : +0.026

```

We may now illustrate the quality of the global ranking of the movies shown with the heat map in Fig. 16.2.

16.5 Fitness of ranking heuristics

We reconsider the bipolar-valued outranking digraph g modelling the pairwise global “*at least as well rated as*” relation among the 25 movies seen above.

Listing 16.6 Global movies outranking digraph

```

1 >>> g = BipolarOutrankingDigraph(t,Normalized=True)
2   ----- Object instance description -----
3   Instance class    : BipolarOutrankingDigraph
4   Instance name     : rel_grafittiPerfTab.xml
5   Actions          : 25
6   Criteria         : 15
7   Size             : 390
8   Determinateness  : 65%
9   Valuation domain : {'min': Decimal('-1.0'),
10                           'med': Decimal('0.0'),
11                           'max': Decimal('1.0'),}

```

```

12 >>> g.computeCoSize()
13     188

```

Out of the $25 \times 24 = 600$ irreflexive movie pairs, digraph g contains 390 positively validated, 188 positively invalidated, and 22 indeterminate outranking situations (see the zero-valued cells in Fig. 16.4).

Let us now compute the normalized majority margin $r(\Leftrightarrow)$ of the equivalence between the marginal critic's pairwise ratings and the global NETFLOWS ranking shown in the ordered heat map (see Fig. 16.2).

Listing 16.7 Computing marginal criterion correlations with global NETFLOWS ranking

```

1 >>> from linearOrders import NetFlowsOrder
2 >>> nf = NetFlowsOrder(g)
3 >>> nf.netFlowsRanking
4     ['mv_QS', 'mv_RR', 'mv_DG', 'mv_NP', 'mv_HN', 'mv_HS', 'mv_SM',
5      'mv_JB', 'mv_PE', 'mv_FC', 'mv_TP', 'mv_CM', 'mv_DF', 'mv_TM',
6      'mv_DJ', 'mv_AL', 'mv_RG', 'mv_MB', 'mv_GH', 'mv_HP', 'mv_BI',
7      'mv_DI', 'mv_FF', 'mv_GG', 'mv_TF']
8 >>> for i,item in enumerate():
9 ...     g.computeMarginalVersusGlobalRankingCorrelations(\n
10 ...         nf.netFlowsRanking,ValuedCorrelation=True) :\n
11 ...     print('r(%s<=>nf) = %+.3f' % (item[1],item[0]) )
12
13     r(JH<=>nf) = +0.500
14     r(JPT<=>nf) = +0.430
15     r(AP<=>nf) = +0.323
16     r(DR<=>nf) = +0.263
17     r(MR<=>nf) = +0.247
18     r(VT<=>nf) = +0.227
19     r(GS<=>nf) = +0.160
20     r(CS<=>nf) = +0.140
21     r(SJ<=>nf) = +0.137
22     r(RR<=>nf) = +0.133
23     r(TD<=>nf) = +0.110
24     r(CF<=>nf) = +0.110
25     r(SF<=>nf) = +0.103
26     r(AS<=>nf) = +0.080
27     r(FG<=>nf) = +0.027

```

In Listing 16.7 (see Lines 13-27), we recover the relational equivalence characteristic values shown in the third row of the table in Fig. ???. The global NETFLOWS ranking represents obviously a rather balanced compromise with respect to all movie critics' opinions as there appears no valued negative correlation with anyone of them. The NETFLOWS ranking apparently takes also correctly in 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 digraph g may be furthermore computed as follows:

Listing 16.8 Computing correlation between NETFLOWS and global outranking

```

1 >>> corrgnf = g.computeOrdinalCorrelation(nf)
2 >>> g.showCorrelation(corrgnf)

```

```

3 Correlation indexes:
4 Extended Kendall tau      : +0.780
5 Epistemic determination   :  0.300
6 Bipolar-valued equivalence : +0.234

```

We notice in Listing ?? Line 4 that the ordinal correlation $\tau(g, nf)$ index between the NETFLOWS ranking nf and the determined part of the outranking digraph g is quite high (+0.78). Due to the rather high number of missing data, the r -valued relational equivalence between the nf and the g digraph, with a characteristics value of only +0.234, may be misleading. Yet, +0.234 still corresponds to a 62% majority support of the movie critics' rating opinions.

It would be interesting to compare similarly the correlations one may obtain with other global ranking heuristics, like the COPELAND.

16.6 Illustrating preference divergences

The valued relational equivalence index gives us a further measure for studying how *divergent* appear the rating opinions expressed by the movie critics.

	AP	AS	CF	CS	DR	FG	GS	JH	JPT	MR	RR	SF	SJ	TD	VT
AP	+0.63	+0.04	+0.19	+0.09	+0.22	-0.01	+0.11	+0.23	+0.25	+0.08	+0.02	+0.04	+0.19	+0.04	+0.12
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
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
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
DR					+0.45	+0.03	+0.07	+0.17	+0.23	+0.16	+0.06	+0.03	+0.10	+0.07	+0.10
FG						+0.15	-0.01	+0.04	+0.01	+0.06	-0.09	+0.02	+0.01	+0.01	+0.02
GS							+0.40	+0.07	+0.07	+0.09	-0.02	+0.00	+0.06	+0.04	+0.04
JH								+0.77	+0.28	+0.26	+0.15	+0.12	+0.10	+0.05	+0.14
JPT									+0.92	+0.15	+0.06	+0.09	+0.08	+0.08	+0.17
MR										+0.63	+0.10	+0.08	+0.03	+0.09	+0.10
RR											+0.51	+0.04	+0.01	+0.05	+0.05
SF												+0.18	+0.01	+0.02	+0.05
SJ													+0.51	+0.03	+0.07
TD														+0.26	+0.00
VT															+0.40

Fig. 16.6 Pairwise valued correlation of movie critics.

It is remarkable to notice here that, due to the quite numerous missing data, all pairwise valued ordinal correlation indexes $r(x \leftrightarrow y)$ appear to be of low value, except the diagonal ones. These reflexive indexes $r(x \leftrightarrow x)$ would trivially all amount to +1.0 in a plainly determined case. Here they indicate a reflexive normalized determination score d , i.e. the proportion of pairs of movies each critic did evaluate. Critic 'JPT' (the editor of the Graffiti magazine), for instance, evaluated all but one ($d = 24 \times 23 / 600 = 0.92$), whereas critic 'FG' evaluated only 10 movies among the 25 in discussion ($d = 10 \times 9 / 600 = 0.15$).

To get a picture of the actual divergence of rating opinions concerning jointly seen pairs of movies, we may develop a Principal Component Analysis Footnote[2]

of the corresponding τ correlation matrix. The 3D plot of the first 3 principal axes is shown in Fig. 16.2.

```
>>> g.export3DplotOfCriteriaCorrelation(ValuedCorrelation=False)
```

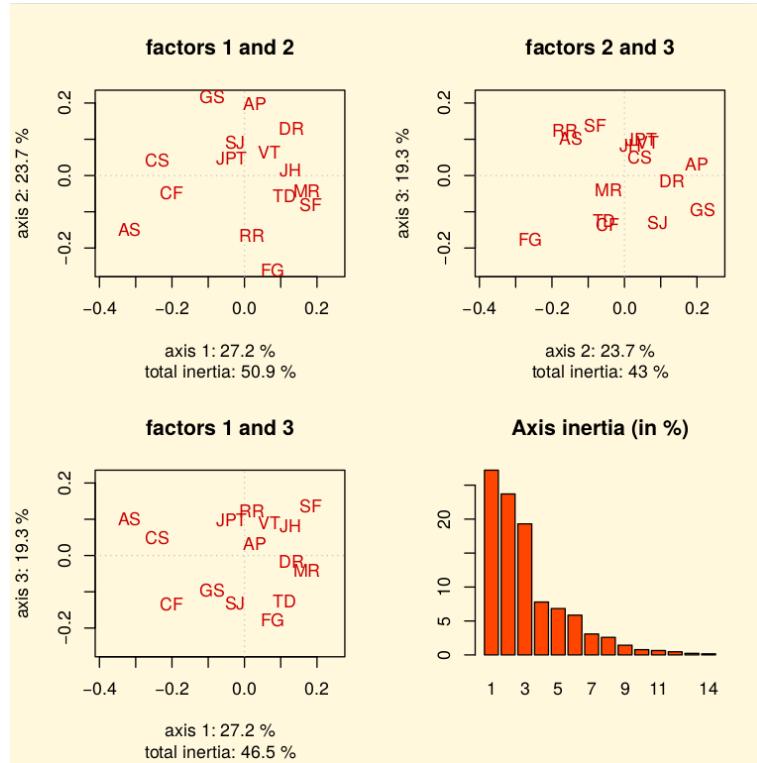


Fig. 16.7 3D PCA plot of the criteria ordinal correlation matrix.

The first 3 principal axes support together about 70% of the total inertia. Most eccentric and opposed in their respective rating opinions appear, on the first principal axis with 27.2% inertia, the conservative daily press against labour and public press. On the second principal axis with 23.7% inertia, it is the people press versus the cultural critical press. And, on the third axis with still 19.3% inertia, the written media appear most opposed to the radio media.

16.7 Exploring the “better rated” and the “as well as rated” opinions

In order to furthermore study the quality of a ranking result, it may be interesting to have a separate view on the asymmetric and symmetric parts of the “*at least as well rated as*” opinions (see Section ??).

Let us first have a look at the pairwise asymmetric part, namely the “*better rated than*” and “*less well rated than*” opinions of the movie critics.

```

1 >>> from digraphs import AsymmetricPartialDigraph
2 >>> ag = AsymmetricPartialDigraph(g)
3 >>> ag.showHTMLRelationTable()
4 ...     actionsList=g.computeNetFlowsRanking(),ndigits=0)
```

Valued Adjacency Matrix

mv_SS	mv_QS	mv_RR	mv_DG	mv_NP	mv_HN	mv_HS	mv_SM	mv_JB	mv_PE	mv_FC	mv_TP	mv_CM	mv_DF	mv_TM	mv_DJ	mv_AI	mv_RG	mv_MB	mv_GH	mv_HP	mv_BI	mv_DL	mv_FF	mv_GG	mv_TF	
mv_QS	-	11	12	11	10	9	14	9	11	13	9	10	9	9	7	6	9	9	7	6	5	9	15	8	8	
mv_RR	-5	-	0	0	0	0	0	5	0	0	0	9	12	9	10	8	9	10	8	9	6	10	13	10	9	
mv_DG	-8	0	-	0	0	0	0	0	0	0	7	7	10	10	6	8	10	7	9	5	6	7	9	9	9	
mv_NP	-7	0	0	-	0	0	0	0	0	8	0	0	8	0	8	7	6	4	6	8	6	5	7	12	10	7
mv_HN	-10	0	0	0	-	0	0	0	0	0	0	0	0	0	0	10	7	6	8	8	5	7	13	9	11	
mv_HS	-3	0	0	0	0	0	-	0	0	0	0	0	5	0	7	0	0	10	9	8	9	7	11	14	9	11
mv_SM	-8	0	0	0	0	0	0	-	0	0	0	12	9	0	6	0	9	9	10	7	9	7	11	15	11	11
mv_JB	-9	-1	0	0	0	0	0	0	-	0	0	0	6	8	0	9	9	8	8	5	11	15	12	8	8	
mv_PE	-7	0	0	0	0	0	0	0	-	0	4	10	0	0	6	8	8	0	6	5	9	13	9	8	8	
mv_FC	-9	0	0	0	0	0	0	0	0	-	10	6	0	5	3	8	6	9	9	5	7	10	12	10	11	
mv_TP	-7	0	-1	0	0	0	0	-4	-4	0	-	0	0	6	4	0	7	6	7	7	9	14	12	10	10	
mv_CM	-8	-1	-5	-4	-3	-3	-1	0	-2	-2	0	-	0	0	8	0	10	0	0	7	3	9	14	11	8	
mv_DF	-9	-2	-2	0	0	0	0	0	0	0	0	-	4	6	-1	6	8	6	0	6	8	0	0	9	9	
mv_TM	-3	-7	-6	-2	-2	-1	0	0	0	-1	-2	0	-2	-	0	0	0	9	7	5	7	9	5	8	8	
mv_DL	-7	-8	-4	-3	0	4	0	0	-2	-1	0	0	0	0	-	0	0	4	3	6	4	4	0	0	5	
mv_AL	-4	0	-4	-6	-6	0	-1	-3	-4	-2	0	0	3	0	0	-	0	1	0	0	5	3	3	3	3	
mv_RG	-7	-6	-4	-4	-1	-2	-1	-5	-4	-2	-3	-2	-2	0	0	0	0	0	0	4	0	0	4	9	9	
mv_MB	-9	-8	-5	-4	-4	-9	-4	-1	-2	-1	-2	0	-2	0	0	-	0	4	4	4	2	6	0	0	8	
mv_GH	-5	-8	-7	-6	-8	-4	-3	0	0	-5	-5	0	-2	-1	-3	0	3	0	0	-	5	0	0	0	3	
mv_HP	-4	-5	-5	-2	-4	-3	-3	-4	-4	-3	-1	-3	0	-7	-2	-1	0	-2	-1	-	0	0	0	7	8	
mv_BI	-5	-4	-6	-1	-1	-7	-5	-5	-5	-3	-1	-3	0	-3	-2	-0	0	0	0	0	-	0	0	-1	0	
mv_DL	-9	-6	-5	-5	-3	-3	-3	-7	-7	-6	-1	-7	0	-5	0	0	0	0	0	0	-	0	4	0	0	
mv_FF	-11	-11	-7	-10	-5	0	-3	-9	-9	-8	-2	-6	0	-1	0	-1	0	0	0	0	5	0	-	0	11	
mv_GG	-6	-8	-7	-4	-5	-7	-9	-10	-9	-4	-2	-7	0	-3	-6	-3	-1	-1	-4	-3	-2	-1	-3	5	-2	0
mv_TF	-8	-7	-7	-5	-7	-11	-9	-8	-8	-7	-6	-8	-3	-6	-3	-1	-1	-4	-3	0	0	0	-1	0	-	-

Valuation domain: [-19.00; +19.00]

Fig. 16.8 Asymmetric part of *graffiti07* digraph

We notice here that the NETFLOWS ranking rule inverts in fact just three “*less well rated than*” opinions and four “*better rated than*” ones. A similar look at the symmetric part, the pairwise “*as well rated as*” opinions, suggests a preordered preference structure in several equivalently rated classes.

```

1 >>> from digraphs import SymmetricPartialDigraph
2 >>> sg = SymmetricPartialDigraph(g)
3 >>> sg.showHTMLRelationTable()
4 ...     actionsList=g.computeNetFlowsRanking(), \
5 ...     ndigits=0)
```

Such a preordering of the movies may, for instance, be computed with the `computeRankingByChoosing()` method, where we iteratively extract dominant kernels –best remaining choices– and absorbent kernels –worst remaining choices– (see the next Chapter). We operate therefore on the asymmetric “*better rated than*” opinions, i.e. the codual (Footnote[3]) of the “*at least as well rated as*” opinions (see Listing 16.9 Line 2).

Valued Adjacency Matrix

mv_QS	mv_RS	mv_RR	mv_DG	mv_NP	mv_HN	mv_HS	mv_SM	mv_JB	mv_PF	mv_FC	mv_TP	mv_CM	mv_DF	mv_TM	mv_DJ	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_RR	0	-	5	7	8	6	4	0	2	9	10	0	0	0	0	8	0	0	0	0	0	0	0	0	0
mv_DG	0	9	-	9	10	5	9	7	8	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
mv_NP	0	5	7	-	10	3	7	9	8	11	9	0	12	0	0	0	0	0	0	0	0	0	0	0	0
mv_HN	0	4	8	8	-	5	9	9	8	10	10	0	10	0	8	0	0	0	0	0	0	0	0	0	0
mv_HS	0	2	5	5	3	-	10	2	5	6	9	0	10	0	0	1	0	0	0	0	0	0	0	0	0
mv_SM	0	6	7	5	7	6	-	6	6	10	12	0	0	10	6	4	0	0	0	0	0	0	0	0	0
mv_JB	0	0	5	1	3	4	8	-	9	6	0	13	6	8	9	0	0	0	8	0	0	0	0	0	0
mv_PF	0	2	6	0	4	3	6	11	-	5	0	0	5	7	0	0	0	0	5	0	0	0	0	0	0
mv_FC	0	5	11	9	8	2	8	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	0
mv_CM	0	0	0	0	0	0	0	0	0	5	0	0	4	-	3	8	9	0	7	5	0	0	0	0	0
mv_DF	0	0	0	0	2	2	2	0	1	1	5	1	-	0	6	0	0	0	0	5	6	8	15	13	0
mv_TM	0	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	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	0
mv_RG	0	0	0	0	0	0	0	0	0	0	0	0	1	3	4	-	1	0	0	0	8	9	0	0	0
mv_MB	0	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	1	0	0	0	0	6	-	0	2	5	3	0	0
mv_HP	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	2	0	0	-	3	6	5	0	8
mv_BI	0	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	0	0	1	4	0	1	8	5	-	6	0
mv_FF	0	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	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	0	2	2	0	0	2	-

Valuation domain: [-19.00; +19.00]

Fig. 16.9 Symmetric part of *graffiti07* digraph**Listing 16.9** Bipolar ranking-by-choosing the Graffiti movies

```

1 >>> from transitiveDigraphs import RankingByChoosingDigraph
2 >>> rbc = RankingByChoosingDigraph(g, CoDual=True)
3 >>> rbc.showRankingByChoosing()
4   Ranking by Choosing and Rejecting
5     1st Best Choice ['mv_QS']
6     2nd Best Choice ['mv_DG', 'mv_FC', 'mv_HN', 'mv_HS', 'mv_NP',
7                           'mv_PF', 'mv_RR', 'mv_SM']
8     3rd Best Choice ['mv_CM', 'mv_JB', 'mv_TM']
9     4th Best Choice ['mv_AL', 'mv_TP']
10    4th Worst Choice ['mv_AL', 'mv_TP']
11    3rd Worst Choice ['mv_GH', 'mv_MB', 'mv_RG']
12    2nd Worst Choice ['mv_DF', 'mv_DJ', 'mv_FF', 'mv_GG']
13    1st Worst Choice ['mv_BI', 'mv_DI', 'mv_HP', 'mv_TF']

```

In the next Chapter 17, we thoroughly discuss the computation of kernels in bipolar-valued digraphs.

Chapter 17

On computing digraph kernels

Abstract

17.1 What is a graph kernel ?

We call *choice* in a graph, respectively a digraph, a subset of its vertices, resp. 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. 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.

In all graph or symmetric digraph, 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, then 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.

We may illustrate this coincidence between MISs and kernels in graphs and symmetric digraphs with the following random 3-regular graph instance.

```
1 >>> from graphs import RandomRegularGraph
2 >>> g = RandomRegularGraph(order=12,\n3     ...                           degree=3, seed=100)
4 >>> g.exportGraphViz('random3RegularGraph')
5     *---- exporting a dot file for GraphViz tools --*
6     Exporting to random3RegularGraph.dot
7     fdp -Tpng random3RegularGraph.dot\
8           -o random3RegularGraph.png
```

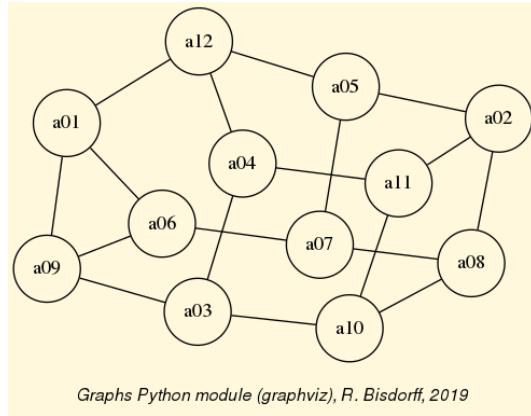


Fig. 17.1 A random 3-regular graph instance

A random MIS in this graph may be computed for instance by using the `MISModel` class.

```

1 >>> from graphs import MISModel
2 >>> mg = MISModel(g)
3   Iteration: 1
4     Running a Gibbs Sampler for 660 step !
5     {'a06', 'a02', 'a12', 'a10'} is maximal !
6 >>> mg.exportGraphViz('random3RegularGraph_mis')
7   *---- exporting a dot file for GraphViz tools ----*
8   Exporting to random3RegularGraph-mis.dot
9   fdp -Tpng random3RegularGraph-mis.dot \
10      -o random3RegularGraph-mis.png

```

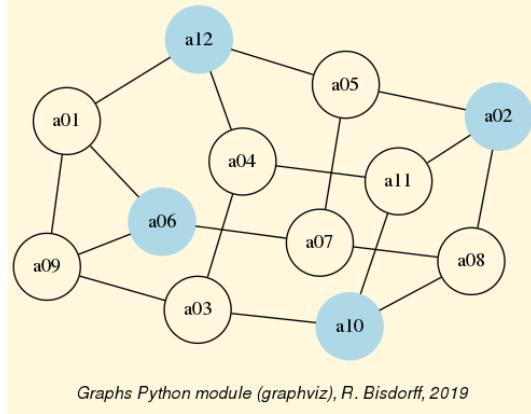


Fig. 17.2 A random MIS colored in the random 3-regular graph

It is easily verified in Fig. 17.2 above, 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.

```

1 >>> g.showMIS()
2     *--- Maximal Independent Sets ---*
3     ['a01', 'a02', 'a03', 'a07']
4     ['a01', 'a04', 'a05', 'a08']
5     ['a04', 'a05', 'a08', 'a09']
6     ['a01', 'a04', 'a05', 'a10']
7     ['a04', 'a05', 'a09', 'a10']
8     ['a02', 'a03', 'a07', 'a12']
9     ['a01', 'a03', 'a07', 'a11']
10    ['a05', 'a08', 'a09', 'a11']
11    ['a03', 'a07', 'a11', 'a12']
12    ['a07', 'a09', 'a11', 'a12']
13    ['a08', 'a09', 'a11', 'a12']
14    ['a04', 'a05', 'a06', 'a08']
15    ['a04', 'a05', 'a06', 'a10']
16    ['a02', 'a04', 'a06', 'a10']
17    ['a02', 'a03', 'a06', 'a12']
18    ['a02', 'a06', 'a10', 'a12']
19    ['a01', 'a02', 'a04', 'a07', 'a10']
20    ['a02', 'a04', 'a07', 'a09', 'a10']
21    ['a02', 'a07', 'a09', 'a10', 'a12']
22    ['a01', 'a03', 'a05', 'a08', 'a11']
23    ['a03', 'a05', 'a06', 'a08', 'a11']
24    ['a03', 'a06', 'a08', 'a11', 'a12']
25    number of solutions: 22
26    cardinality distribution
27    card.: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
28    freq.: [0, 0, 0, 0, 16, 6, 0, 0, 0, 0, 0, 0, 0]
29    execution time: 0.00045 sec.
30    Results in self.misset
31 >>> g.misset
32     [frozenset({'a02', 'a01', 'a07', 'a03'}),
33      frozenset({'a04', 'a01', 'a08', 'a05'}),
34      frozenset({'a09', 'a04', 'a08', 'a05'}),
35      ...
36      ...
37      frozenset({'a06', 'a02', 'a12', 'a10'}),
38      frozenset({'a06', 'a11', 'a08', 'a03', 'a05'}),
39      frozenset({'a03', 'a06', 'a11', 'a12', 'a08'})]
```

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.

17.2 Initial and terminal kernels

In an oriented graph context, the internal stability condition of the kernel concept remains untouched; however, the external stability condition gets indeed split up by the orientation into two lateral cases:

1. A *dominant* stability condition, where each non selected node is dominated by at least one member of the kernel;
2. An *absorbent* stability condition, where each non selected node is absorbed by at least one member of the kernel.

A both internally **and** dominant, resp. absorbent stable choice is called a dominant or *initial*, resp. an absorbent or *terminal* kernel. From a topological perspective, the initial kernel concept looks from the outside of the digraph into its interior, whereas the terminal kernel looks from the interior of a digraph toward its outside. From an algebraic perspective, the initial kernel is a prefix operand, and the terminal kernel is a postfix operand in the BERGE kernel equation systems (see Section ??).

Furthermore, as the kernel concept involves conjointly a positive logical refutation (the internal stability) and a positive logical affirmation (the external stability), it appeared rather quickly necessary in our operational developments to adopt a bipolar characteristic $[-1.0, 1.0]$ valuation domain, modelling logical negation by a change of numerical sign and including explicitly a third median logical value (0.0) expressing logical *indeterminateness* (neither positive, nor negative, see [BIS-2000] and [BIS-2004a]).

In such a bipolar-valued context, we call *prekernel* a choice which is *externally stable* and for which the internal stability condition is *valid or indeterminate*. We say that the independence condition is in this case only *weakly* validated. Notice that all kernels are hence prekernels, but not vice-versa.

In graphs or symmetric digraphs, where there is essentially no apparent *laterality*, all prekernels are initial and terminal at the same time. A universal example is given by the *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
15     ['2'] ind. : 1.0; dom. : 1.0; abs. : 1.0
16     ['3'] ind. : 1.0; dom. : 1.0; abs. : 1.0
17     ['4'] ind. : 1.0; dom. : 1.0; abs. : 1.0
18     ['5'] ind. : 1.0; dom. : 1.0; abs. : 1.0
19     Absorbent kernels :
20     ['1'] ind. : 1.0; dom. : 1.0; abs. : 1.0
21     ['2'] ind. : 1.0; dom. : 1.0; abs. : 1.0
22     ['3'] ind. : 1.0; dom. : 1.0; abs. : 1.0
23     ['4'] ind. : 1.0; dom. : 1.0; abs. : 1.0
24     ['5'] ind. : 1.0; dom. : 1.0; abs. : 1.0

```

```

25      *----- statistics -----
26      graph name: complete
27      number of solutions
28      dominant kernels : 5
29      absorbent kernels: 5
30      cardinality frequency distributions
31      cardinality : [0, 1, 2, 3, 4, 5]
32      dominant kernel : [0, 5, 0, 0, 0, 0]
33      absorbent kernel: [0, 5, 0, 0, 0, 0]
34      Execution time : 0.00004 sec.
35      Results in sets: dompreKernels
36      and abspreKernels.

```

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 in two other universal digraph models, the *empty* and the *indeterminate* digraph.

```

1 >>> from digraphs import EmptyDigraph
2 >>> ed = EmptyDigraph(order=5)
3 >>> ed.showPreKernels()
4     *--- Computing preKernels ---*
5     Dominant kernel :
6         ['1', '2', '3', '4', '5']
7             independence : 1.0
8             dominance   : 1.0
9             absorbency  : 1.0
10    Absorbent kernel :
11        ['1', '2', '3', '4', '5']
12            independence : 1.0
13            dominance   : 1.0
14            absorbency  : 1.0

```

In the empty digraph, the whole set of nodes gives indeed at the same time the unique initial and terminal prekernel. Similarly, for the indeterminate digraph.

```

1 >>> from digraphs import IndeterminateDigraph
2 >>> id = IndeterminateDigraph(order=5)
3 >>> id.showPreKernels()
4     *--- Computing preKernels ---*
5     Dominant prekernel :
6         ['1', '2', '3', '4', '5']
7             independence : 0.0
8             dominance   : 1.0
9             absorbency  : 1.0
10    Absorbent prekernel :
11        ['1', '2', '3', '4', '5']
12            independence : 0.0
13            dominance   : 1.0
14            absorbency  : 1.0

```

Both these results make sense, as in a completely empty or indeterminate digraph, there is no *interior* of the digraph defined, only a *border* which is hence at

the same time an initial and terminal prekernel. Notice however, that in the latter indeterminate case, the complete set of nodes verifies only weakly the internal stability condition (see above).

Other common digraph models, although being clearly oriented, may show nevertheless no apparent laterality, like *odd chordless circuits*, i.e. holes surrounded by an oriented cycle -a circuit- of odd length. They do not admit in fact any initial or terminal prekernel.

```
1 >>> from digraphs import CirculantDigraph
2 >>> c5 = CirculantDigraph(order=5,circulants=[1])
3 >>> c5.showPreKernels()
4     *----- statistics -----
5     digraph name: c5
6     number of solutions
7     dominant prekernels : 0
8     absorbent prekernels: 0
```

Chordless circuits of *even* length $2 \times k$, with $k > 1$, contain however two isomorphic prekernels of cardinality k which qualify conjointly as initial and terminal candidates.

```
1 >>> c6 = CirculantDigraph(order=6,circulants=[1])
2 >>> c6.showPreKernels()
3     *--- Computing preKernels ---*
4         Dominant preKernels :
5             ['1', '3', '5'] ind. : 1.0, dom. : 1.0, abs. : 1.0
6             ['2', '4', '6'] ind. : 1.0, dom. : 1.0, abs. : 1.0
7         Absorbent preKernels :
8             ['1', '3', '5'] ind. : 1.0, dom. : 1.0, abs. : 1.0
9             ['2', '4', '6'] ind. : 1.0, dom. : 1.0, abs. : 1.0
```

Chordless circuits of even length may thus be indifferently oriented along two opposite directions. Notice by the way 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.

```
1 >>> dc6 = -c6    # dc6 = DualDigraph(c6)
2 >>> dc6.showPreKernels()
3     *----- statistics -----
4         graph name: dual_c6
5         number of solutions
6             dominant prekernels : 0
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.

The `CirculantDigraph` class provides a parameter for constructing such weak chordless circuits.

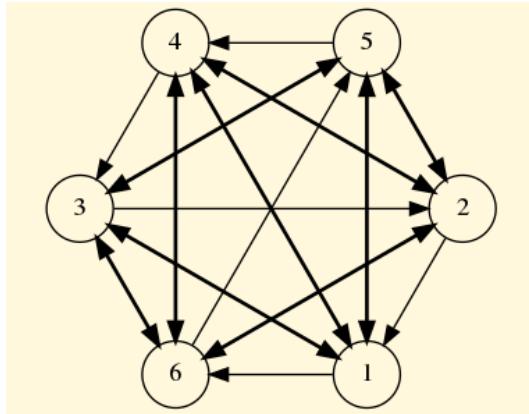


Fig. 17.3 The dual of the chordless 6-circuit

Rubis Python Server (graphviz), R. Bisдорff, 2008

It is worth noticing that the dual version (Footnote[14]) of a weak circuit corresponds to its *converse* version, i.e. $-c6 = \sim c6$ (see Fig. 17.4).

```

1 >>> (-c6).exportGraphViz()
2     *---- exporting a dot file for GraphViz tools -----*
3     Exporting to dual_c6.dot
4     circo -Tpng dual_c6.dot -o dual_c6.png
5 >>> (~c6).exportGraphViz()
6     *---- exporting a dot file for GraphViz tools -----*
7     Exporting to converse_c6.dot
8     circo -Tpng converse_c6.dot -o converse_c6.png

```

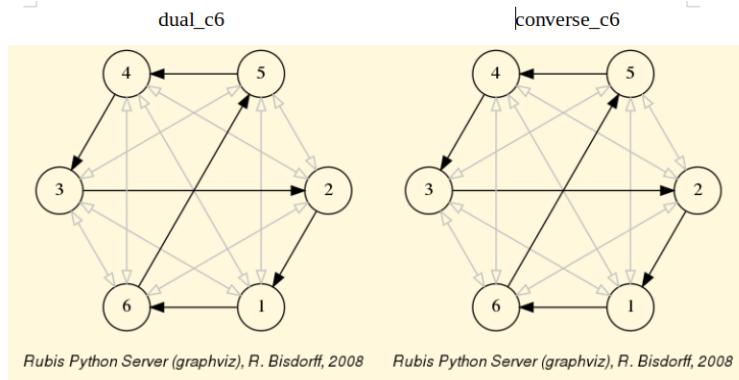


Fig. 17.4 Dual and converse of the weak 6-circuit

It is worthwhile noticing that weak chordless circuits of length n are in fact part of the class of digraphs that are invariant under the *codual* transform, $cn = -(\sim$

$cn) = \sim (-cn)$ Footnote [13]. In the case, now, of an odd weak chordless circuit, neither the weak chordless circuit, nor its dual, converse, or codual versions will admit any initial or terminal 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, resp. last ranked, nodes deliver unique initial, resp. terminal, kernels. Similarly, in finite preorders, the first, resp. last, equivalence classes deliver the unique initial, resp. unique terminal, kernels. More generally, in finite partial orders, i.e. asymmetric and transitive digraphs, topological sort algorithms will easily reveal on the first, resp. last, level all unique initial, resp. terminal, kernels.

In genuine random digraphs, however, we may need to check for each of its MISs, whether one, both, or none of the lateralized external stability conditions may be satisfied. Consider, for instance, the following random digraph instance of order 7 and generated with an arc probability of 30%.

```

1 >>> from randomDigraphs import RandomDigraph
2 >>> rd = RandomDigraph(order=7, arcProbability=0.3, seed=5)
3 >>> rd.exportGraphViz('randomLaterality')
4     *---- exporting a dot file for GraphViz tools ----*
5     Exporting to randomLaterality.dot
6     dot -Grankdir=BT -Tpng randomLaterality.dot\
7             -o randomLaterality.png

```

The random digraph shown in Fig. 17.5 has no apparent special properties, except from being connected (see Line 3 below).

```

1 >>> rd.showComponents()
2     *--- Connected Components ---*
3     1: ['a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7']
4 >>> rd.computeSymmetryDegree(Comments=True, InPercents=True)
5     Symmetry degree (%) of digraph <randomDigraph>:
6     arcs x>y: 14, symmetric: 1, asymmetric: 13
7     symmetric/arcs = 7.1
8 >>> rd.computeChordlessCircuits()
9     [] # no chordless circuits detected
10 >>> rd.computeTransitivityDegree(Comments=True, InPercents=True)
11     Transitivity degree (%) of graph <randomDigraph>:
12     triples x>y>z: 23, closed: 11, open: 12
13     closed/triples = 47.8

```

There are no chordless circuits (see Line 9 above) and more than half of the required transitive closure is missing (see Line 12 above).

Now, we know that its potential prekernels must be among its set of maximal independent choices.

```

1 >>> rd.showMIS()

```

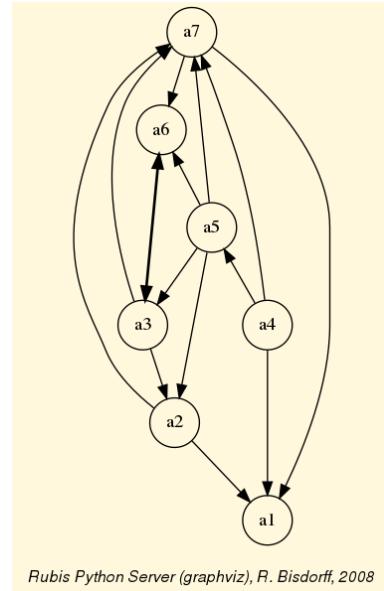


Fig. 17.5 A random digraph instance of order 7 and arc probability 0.3. The digraph instance is neither asymmetric ($a_3 \Leftrightarrow a_6$) nor symmetric ($a_2 \rightarrow a_1, a_1 \not\rightarrow a_2$) and, the digraph is not transitive ($a_5 \rightarrow a_2 \rightarrow a_1$, but $a_5 \not\rightarrow a_1$).

```

2      *--- Maximal independent choices ---*
3      ['a2', 'a4', 'a6']
4      ['a6', 'a1']
5      ['a5', 'a1']
6      ['a3', 'a1']
7      ['a4', 'a3']
8      ['a7']
9  >>> rd.showPreKernels()
10     *--- Computing preKernels ---*
11     Dominant preKernels :
12     ['a2', 'a4', 'a6']
13     independence : 1.0
14     dominance   : 1.0
15     absorbency   : -1.0
16     covering     : 0.500
17     ['a4', 'a3']
18     independence : 1.0
19     dominance   : 1.0
20     absorbency   : -1.0
21     covering     : 0.600
22     Absorbent preKernels :
23     ['a3', 'a1']
24     independence : 1.0
25     dominance   : -1.0
26     absorbency   : 1.0
27     covering     : 0.500
28     ['a6', 'a1']
29     independence : 1.0
30     dominance   : -1.0

```

```

31      absorbency   : 1.0
32      covering    : 0.600

```

Among the six MISs contained in this random digraph (see above Lines 3-8) we discover two initial and two terminal kernels (Lines 12-34). Notice by the way the covering values (between 0.0 and 1.0) shown by the `showPreKernels()` method (Lines 17, 22, 28 and 33). The higher this value, the more the corresponding prekernel candidate makes apparent the digraph's laterality. We may hence redraw the same digraph in Fig. 17.6 by looking into its interior via the best covering initial prekernel candidate: the dominant choice $\{a3', a4'\}$ (coloured in yellow), and looking out of it via the best covered terminal prekernel candidate: the absorbent choice $\{a1', a6'\}$ (coloured in blue).

```

1 >>> rd.exportGraphViz(fileName='orientedLaterality', \
2 ...                     bestChoice=set(['a3', 'a4']), \
3 ...                     worstChoice=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

```

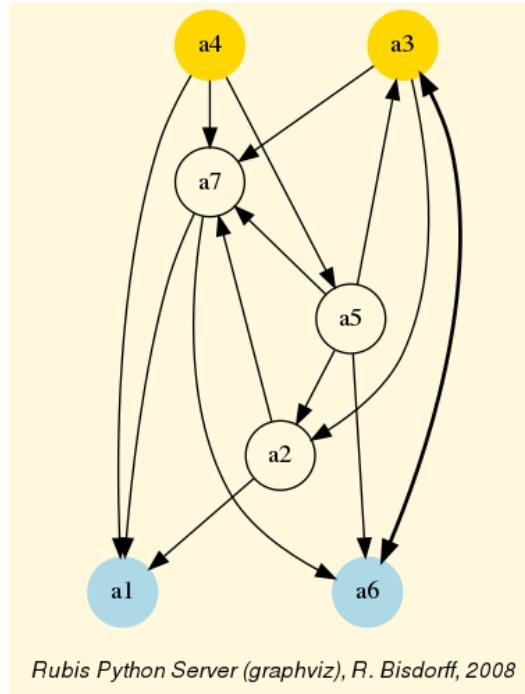


Fig. 17.6 A random digraph oriented by best covering initial and best covered terminal kernel

In Algorithmic Decision Theory, initial and terminal prekernels may provide convincing best, resp. last, choice recommendations.

17.4 Computing first and last choice recommendations

To illustrate this idea, let us finally compute first and last choice recommendations in the following random bipolar-valued outranking digraph.

```

1 >>> from outrankingDigraphs import\
2 ...                 RandomBipolarOutrankingDigraph
3 >>> g = RandomBipolarOutrankingDigraph(seed=5)
4 >>> g
5      ----- Object instance description -----
6      Instance class    : RandomBipolarOutrankingDigraph
7      Instance name     : randomOutranking
8      Actions          : 7
9      Criteria         : 7
10     Size              : 26
11     Determinateness   : 34.275
12     Valuation domain  : {'min': -100.0, 'med': 0.0, 'max': 100.0}
13 >>> g.showHTMLPerformanceTableau()

```

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	NA	NA	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	NA	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. 17.7 The performance tableau of a random outranking digraph instance

The underlying random performance tableau shown in Fig. 17.7 reveals the performance grading of 7 potential decision actions with respect to 7 decision criteria supporting each an increasing performance scale from 0.0 to 100.0. Notice the missing performance data concerning decision actions 'a2' and 'a5'. The resulting *strict outranking* - i.e. a weighted majority supported *better than without considerable counter-performance* - digraph is shown in Fig. 17.9.

```

1 >>> gcd = ~(-g) # Codual: the converse of the negation
2 >>> gcd.exportGraphViz(fileName='tutOutRanking')
3      ----- exporting a dot file for GraphViz tools -----
4      Exporting to tutOutranking.dot
5      dot -Grankdir=BT -Tpng tutOutranking.dot\
6                           -o tutOutranking.png

1 >>> gcd.showPreKernels()
2      ---- Computing preKernels ----*
3      Dominant preKernels :

```

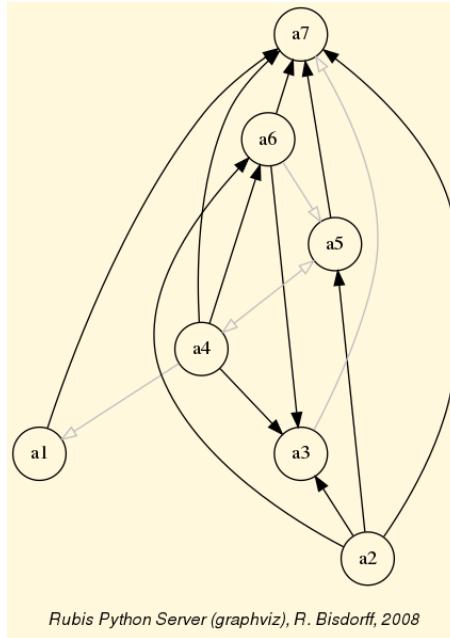


Fig. 17.8 A random strict outranking digraph instance. All decision actions appear strictly better performing than action a_7 . We call it a CONDORCET loser and it is an evident terminal prekernel candidate. On the other side, three actions: a_1 , a_2 and a_4 are not dominated. They give together an initial prekernel candidate.

```

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 Line 4 and 10), the given digraph instance is hence clearly lateralized. Indeed, these initial and terminal prekernels of the codual outranking digraph reveal best, resp. worst, choice recommendations one may formulate on the basis of a given outranking digraph instance.

```

1 >>> g.showBestChoiceRecommendation()
2 Rubis best choice recommendation(s) (BCR)
3 (in decreasing order of determinateness)
4 Credibility domain: [-100.00,100.00]
5 === >> potential best choice(s)
6 * choice           : ['a1', 'a2', 'a4']
7     independence    : 0.00
8     dominance        : 6.98
9     absorbency       : -48.84
10    covering (%)     : 66.67
11    determinateness (%) : 57.97

```

```

12      - most credible action(s) =
13          {'a4': 20.93, 'a2': 20.93}
14  === >> potential worst choice(s)
15      * choice           : ['a3', 'a7']
16      independence       : 0.00
17      dominance          : -74.42
18      absorbency         : 16.28
19      covered (%)        : 80.00
20      determinateness (%) : 64.62
21      - most credible action(s) = { 'a7': 48.84, }

```

Notice that solving the valued BERGE kernel equations provides furthermore a positive characterization of the most credible decision actions in each respective choice recommendation (see Lines 14 and 23 above). Actions 'a2' and 'a4' are equivalent candidates for a unique best choice, and action 'a7' is clearly confirmed as the worst choice.

In Fig. 17.9, we orient the drawing of the strict outranking digraph instance with the help of these best and worst choice recommendations.

```

1 >>> gcd.exportGraphViz(fileName='bestWorstOrientation', \
2 ...                         bestChoice=['a2','a4'], \
3 ...                         worstChoice=['a7'])
4 *---- exporting a dot file for GraphViz tools ----*
5   Exporting to bestWorstOrientation.dot
6   dot -Grankdir=BT -Tpng bestWorstOrientation.dot \
7     -o bestWorstOrientation.png

```

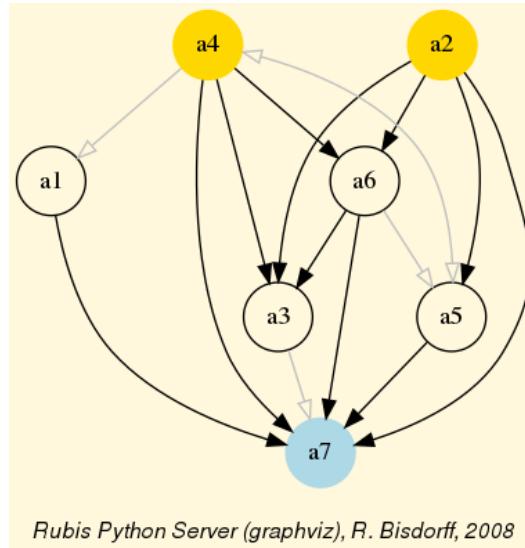


Fig. 17.9 The strict outranking digraph oriented by its best and worst choice recommendations. The gray arrows, like the one between actions a4 and a1, represent indeterminate preferential situations. Action a1 appears hence to be rather incomparable to all the other, except action a7.

It may be interesting to compare this result with a COPELAND ranking of the underlying performance tableau (see Chapter 8 [8](#) on ranking with uncommensurable criteria).

```

1 >>> g.showHTMLPerformanceHeatmap(colorLevels=5, ndigits=0, \
2 ... Correlations=True, rankingRule='Copeland')
```

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. 17.10 Heatmap with Copeland ranking of the performance tableau

In the resulting linear ranking (see Fig. [17.10](#)), action 'a4' is set at first rank, followed by action 'a2'. This makes sense as '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, don't 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. [17.10](#).

17.5 Tractability of prekernel computation

Let us give some hints on the *tractability* of prekernel computations. Detecting all prekernels in a digraph is a famously NP-hard computational 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 $n > 20$ (see [BIS-2006b]). 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 >>> e = EmptyDigraph(order=20)
2 >>> e.showMIS()      # by visiting all  $2^{20}$  independent choices
3     *--- Maximal independent choices ---*
4     [ '1', '2', '3', '4', '5', '6', '7', '8', '9', '10',
5      '11', '12', '13', '14', '15', '16', '17', '18', '19', '20']
6     number of solutions: 1
7     execution time: 1.47640 sec.
8 >>> 2**20
9     1048576

```

Now, there exist more efficient specialized algorithms for directly enumerating MISs and dominant or absorbent kernels contained in specific digraph models without visiting all independent choices (see [BIS-2006b]). *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 7 above and Line 15 below).

```

1 >>> e.showMIS_AH()
2     # by visiting only maximal independent choices
3     *-----*
4     * Python implementation of Hertz's *
5     * algorithm for generating all MISs *
6     * R.B. version 7(6)-25-Apr-2006   *
7     *-----*
8     =====>>> Initial solution :
9     [ '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11',
10      '12', '13', '14', '15', '16', '17', '18', '19', '20']
11     *--- results ---*
12     [ '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11',
13      '12', '13', '14', '15', '16', '17', '18', '19', '20']
14     *--- statistics ---*
15     MIS solutions : 1
16     execution time : 0.00026 sec.
17     iteration history: 1

```

For more or less dense strict outranking digraphs of modest order, as facing usually in Algorithmic Decision Theory applications, enumerating all independent choices remains however in most cases tractable, especially by using a very efficient Python generator (see the `independentChoices()` method below).

```

1 def independentChoices(self,U):
2     """
3         Generator for all independent choices with associated
4         dominated, absorbed and independent neighborhoods
5         of digraph instance self.
6         Initiate with U = self.singletons().
7         Yields [(independent choice, domnb, absnb, indnb)].
8     """
9     if U == []:
10         yield [(frozenset(), set(), set(), set(self.actions)) ]
11     else:
12         x = list(U.pop())
13         for S in self.independentChoices(U):

```

```

14         yield S
15     if x[0] <= S[0][3]:
16         Sxgamdom = S[0][1] | x[1]
17         Sxgamabs = S[0][2] | x[2]
18         Sxindep = S[0][3] & x[3]
19         Sxchoice = S[0][0] | x[0]
20         Sx = [(Sxchoice, Sxgamdom, Sxgamabs, Sxindep)]
21         yield Sx

```

And, checking maximality of independent choices via the external stability conditions during their enumeration with the `computePreKernels()` method (see below) provides the effective advantage of computing all initial and terminal prekernels in a single loop (see Line 10 and [BIS-2006b]).

Listing 17.1 Computing dominant and absorbent preKernels

```

1 def computePreKernels(self):
2     """
3         computing dominant and absorbent preKernels:
4         Result in self.dompreKernels and self.abspreKernels
5     """
6     actions = set(self.actions)
7     n = len(actions)
8     dompreKernels = set()
9     abspreKernels = set()
10    for choice in self.independentChoices(self.singletons()):
11        restactions = actions - choice[0][0]
12        if restactions <= choice[0][1]:
13            dompreKernels.add(choice[0][0])
14        if restactions <= choice[0][2]:
15            abspreKernels.add(choice[0][0])
16    self.dompreKernels = dompreKernels
17    self.abspreKernels = abspreKernels

```

Finally, we use our bipolar epistemic logic framework for computing the credibility that an individual node of the digraph fits with being a member of an initial or terminal prekerenel. For this purpose we make use of the BERGE kernel equation systems Footnote[x][BER-1958p].

17.6 BERGE kernel equation systems

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: $Y \circ R = -Y$, where for all x in X ,

$$(Y \circ R)(x) = \max_{y \in X, x \neq y} (\min(Y(x), R(x, y))), \quad (17.1)$$

then Y characterises an *initialkernel* ([BER-1958p]).

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*.

Let us verify this result 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     independence : 1.0
15     dominance   : 1.0
16     absorbency   : -1.0
17     covering     : 1.000
18     Absorbent preKernels :
19     ['a2']
20     independence : 1.0
21     dominance   : -1.0
22     absorbency   : 1.0
23     covered      : 1.000

```

It is easy to verify that the characteristic vector $[-1, -1, +1]$ satisfies the initial kernel equation system; 'a3' gives an *initial kernel*. Similarly, the characteristic vector $[-1, +1, -1]$ verifies indeed the terminal kernel equation system and hence 'a2' gives a *terminal kernel*.

We succeeded now in generalizing BERGE's kernel equation systems to genuine bipolar-valued digraphs ([BIS-2006-1p]). 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,

$$T(Y) := -(Y \circ R) = Y, \quad (17.2)$$

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 bipolar-valued fixpoint equation:

$$T^2(Y) := -(-(Y \circ R) \circ R) = Y. \quad (17.3)$$

will admit a stable high and a stable low fixpoint solution that converge both to Y_k ([SCH-1985p]).

Inspired by this crisp double fixpoint equation, we observed that for a given bipolar-valued digraph $G(X, R)$, each 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 kernel (see [BIS-2006-2p]).

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 hence the associated bipolar-valued kernel characteristic vectors Y_i in polynomial complexity.

Algorithm

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 17.1);
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 17.3 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.

```

1 >>> from outrankingDigraphs import \
2         RandomBipolarOutrankingDigraph
3 >>> g = RandomBipolarOutrankingDigraph(seed=5)
4 >>> g
5 *----- Object instance description -----*
6 Instance class      : RandomBipolarOutrankingDigraph
7 Instance name       : rel_randomperftab
8 Actions             : 7
9 Criteria            : 7
10 Size                : 26
11 Determinateness (%) : 67.14
12 Valuation domain   : [-1.0;1.0]
13 Attributes          : ['name', 'actions', 'criteria',
14                           'evaluation', 'relation',
15                           'valuationdomain', 'order',
16                           'gamma', 'notGamma']
```

The random outranking digraph g , we consider above for illustration, models the pairwise outranking situations between seven decision alternatives evaluated on seven incommensurable performance criteria. We compute 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
7       dominance   : +0.070
8       absorbency  : -0.488
9       covering    : +0.667
10  Absorbent prekernels :
11    ['a7', 'a3']
12      independence : +0.000
13      dominance   : -0.744
14      absorbency  : +0.163
15      covered     : +0.800
16  *----- statistics -----
17  graph name: converse-dual_rel_randomperftab
18  number of solutions
19  dominant kernels : 1
20  absorbent kernels: 1
21  cardinality frequency distributions
22  cardinality   : [0, 1, 2, 3, 4, 5, 6, 7]
23  dominant kernel : [0, 0, 0, 1, 0, 0, 0, 0]
24  absorbent kernel: [0, 0, 1, 0, 0, 0, 0, 0]
25  Execution time  : 0.00022 sec.

```

The codual outranking digraph, modelling a strict outranking relation, admits an initial prekernel ['a1', 'a2', 'a4'] and a terminal one ['a3', 'a7'] (see Lines 5 and 11).

We may now compute the initial prekernel restricted adjacency table with the `domkernelrestrict()` method.

```

1 >>> k1Relation = gcd.domkernelrestrict(['a1','a2','a4'])
2 >>> gcd.showHTMLRelationTable(
3 ...     actionsList=['a1','a2','a4','a3','a5','a6','a7'], \
4 ...     relation=k1Relation, \
5 ...     tableTitle='K1 restricted adjacency table')

```

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]

Fig. 17.11 Initial kernel ['a1','a2','a4'] restricted adjacency table. This initial prekernel is indeed only weakly independent: The outranking situation between 'a4' and 'a1' is indeed *indeterminate*.

The corresponding initial prekernel membership characteristic vector may be computed with the `computeKernelVector()` method.

Listing 17.2]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,-1.00]
5 initial high vector: [+1.00,+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,+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

```

We start in Listing 17.2 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 now compute the restricted adjacency table for the outranked, i.e. the *terminal* prekernel ['a3','a7'].

```

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')

```

The corresponding bipolar-valued characteristic vector Y_2 may be computed as follows.

```

1 >>>
2     gcd.computeKernelVector(['a3','a7'],Initial=False,Comments=True)
3 --> Terminal prekernel: {'a3', 'a7'}
4 initial low vector :
5     [-1.00,-1.00,-1.00,-1.00,-1.00,-1.00]
4 initial high vector :
5     [+1.00,+1.00,+1.00,+1.00,+1.00,+1.00]

```

K2 restricted adjacency table

r(x 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]

Fig. 17.12 Terminal prekernel [‘a3’, ‘a7’] restricted adjacency table. Again, we notice that this terminal prekernel is indeed only weakly independent.

```

5   1st low vector      : [-0.16,-0.49,
6     0.00,-0.58,-0.16,-0.30,+0.49]
7   1st high vector    :
8     [+1.00,+1.00,+1.00,+1.00,+1.00,+1.00,+1.00]
9   2nd low vector     : [-0.16,-0.49,
10   0.00,-0.58,-0.16,-0.30,+0.49]
11   2nd high vector   : [-0.16,-0.49,
12   0.00,-0.49,-0.16,-0.26,+0.49]
13   3rd low vector    : [-0.16,-0.49,
14   0.00,-0.49,-0.16,-0.26,+0.49]
15   3rd high vector   : [-0.16,-0.49,
16   0.00,-0.49,-0.16,-0.26,+0.49]
17   Iterations        : 3
18   high & low fusion  : [-0.16,-0.49,
19   0.00,-0.49,-0.16,-0.26,+0.49]
20   Choice vector for terminal prekernel: {'a3','a7'}
    'a7': +0.49
    'a3':  0.00
    'a1': -0.16
    'a5': -0.16
    'a6': -0.26
    'a2': -0.49
    'a4': -0.49

```

A unique stable bipolar-valued high and low fixpoint is attained at the third iteration with ‘7’ positively confirmed (about 75% criteria significance majority) as member of this terminal prekernel, whereas the membership of ‘a3’ in this prekernel appears indeterminate. All the remaining nodes have negative membership characteristic values and are hence positively excluded from this prekernel.

When we reconsider the graphviz drawing of this outranking digraph in Fig. 17.9, 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 (see for Listing ??).

To illustrate finally why sometimes we need to operate an epistemic disjunctive fusion of unequal stable low and high membership characteristics vectors (see Step 2.c.), let us consider, for instance, the following crisp 7-cycle graph.

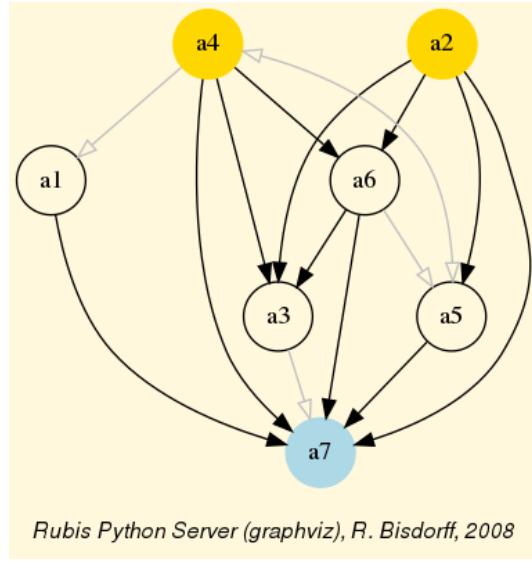


Fig. 17.13 The strict outranking digraph oriented by its initial and terminal prekernels. It becomes obvious here why alternative ‘a1’ is *neither included nor excluded* from the initial prekernel. Same observation is applicable to alternative ‘a3’ which can *neither be included nor excluded* from the terminal prekernel.

```

1 >>> g = CirculantDigraph(order=7,circulants=[-1,1])
2 >>> g
3     *----- Digraph instance description -----*
4     Instance class      : CirculantDigraph
5     Instance name       : c7
6     Digraph Order       : 7
7     Digraph Size        : 14
8     Valuation domain   : [-1.00;1.00]
9     Determinateness (%) : 100.00
10    Attributes          : ['name','order','circulants',
11                           'actions','valuationdomain',
12                           'relation','gamma','notGamma']

```

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. We may compute the corresponding initial kernel characteristic vector.

```

1 >>> g.computeKernelVector(['2','5','7'], \
2                               Initial=True,Comments=True)
3 --> Initial kernel: {'2', '5', '7'}
4 initial low vector : [-1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0]
5 initial high vector: [+1.0, +1.0, +1.0, +1.0, +1.0, +1.0, +1.0]
6 1 st low vector   : [-1.0,  0.0, -1.0, -1.0,  0.0, -1.0,  0.0]
7 1 st high vector  : [+1.0, +1.0, +1.0, +1.0, +1.0, +1.0, +1.0]
8 2 nd low vector   : [-1.0,  0.0, -1.0, -1.0,  0.0, -1.0,  0.0]
9 2 nd high vector  : [ 0.0, +1.0,  0.0,  0.0, +1.0,  0.0, +1.0]
10 stable low vector : [-1.0,  0.0, -1.0, -1.0,  0.0, -1.0,  0.0]
11 stable high vector: [ 0.0, +1.0,  0.0,  0.0, +1.0,  0.0, +1.0]
12 Iterations         : 3
13 low & high fusion  : [-1.0, +1.0, -1.0, -1.0, +1.0, -1.0, +1.0]
14 Choice vector for initial prekernel: {'2', '5', '7'}

```

```

15      '7': +1.00
16      '5': +1.00
17      '2': +1.00
18      '6': -1.00
19      '4': -1.00
20      '3': -1.00
21      '1': -1.00

```

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 transposition 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 numerical algorithms without a possible usage of the number 0. Infinitely many trivial impossibility theorems and dubious logical results come up.

Chapter 18

On confident outrankings with uncertain criteria significances

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? To answer this question, one usually requires *qualified* majority margins for confirming outranking situations. But how to choose such a qualifying majority level: two third, three fourth of the significances? 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, respectively negative.

18.1 Modelling uncertain criteria significances

Let us consider the significance weights of a family F of m criteria to be independent random variables W_j , distributing the potential significance weights 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 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:

- 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 $W_j \sim Beta(4, 4) \times 2E(W_j)$, $V(W_j) = 1/9(E(W_j))^2$.

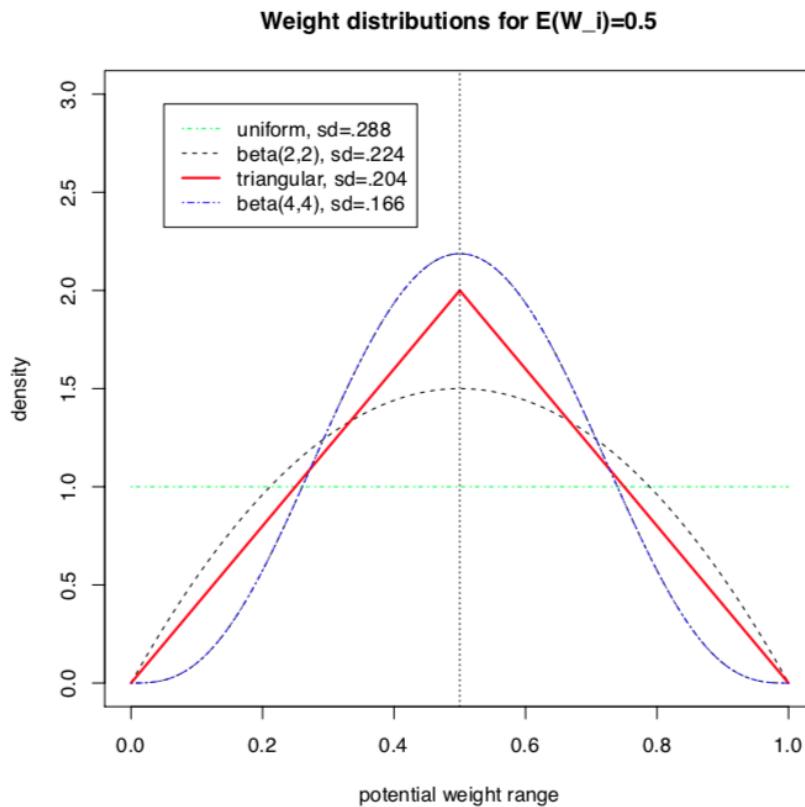


Fig. 18.1 Four models of uncertain significance weights

It is worthwhile noticing 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 $E(W_j)$ (see Fig. 18.1).

18.2 Bipolar-valued likelihood of “at least as good as” 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 :

$$r(x \geq_j y) = \begin{cases} +1 & \text{if } x_j - y_j \leq -ind_j, \\ -1 & \text{if } x_j - y_j \leq -pr_j, \\ 0 & \text{otherwise.} \end{cases} \quad (18.1)$$

Semantics of the marginal bipolar-valued characteristic function:

- $+1$ signifies x is performing at least as good as y on criterion j ,
- -1 signifies that x is not performing at least as good as y on criterion j ,
- 0 signifies that it is unclear whether, on criterion j , x is performing at least as good as y .

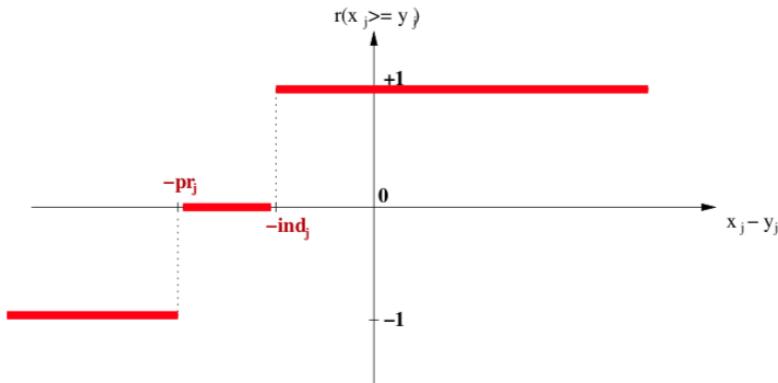


Fig. 18.2 Bipolar-valued outranking characteristic function.

Each criterion j in F contributes the random significance W_j of his “at least as good 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 performing at least as good as y , $\tilde{r}(x \geq y) < 0$ signifies that x is not globally performing at least as good as y , and $\tilde{r}(x \geq y) = 0$ signifies that it is unclear whether x is globally performing at least as good 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 good 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, resp. $P(Y < 0)$ 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:

$$lh(x \geq y) = -\operatorname{erf}\left(\frac{1}{\sqrt{2}} \frac{-E(Y)}{\sqrt{V(Y)}}\right) \quad (18.5)$$

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 good as”* situation. A likelihood of $+1.0$ (resp. -1.0) means the corresponding preferential situation appears *certainly validated* (resp. invalidated).

Example 18.1. Let x and y be evaluated wrt 7 equisignificant criteria; Four criteria positively support that x is *as least as good performing* than y and three criteria support that x is *not at least as good* performing than 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 good 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 good* performing situation becomes: $lh(x \geq y) = 0.66$, which corresponds to a global support of $(0.66 + 1.0)/2 = 83\%$ of the criteria significance weights.

A *Monte Carlo* simulation with 10 000 runs empirically confirms the effective convergence to a Gaussian (see Fig. 18.3 realised with *gretl* Footnote[4]). Indeed, $\tilde{r}(x \geq y) \sim Y = \mathcal{N}(1.03, 1.089)$, with an empirical probability of observing a negative majority margin of about 17%.

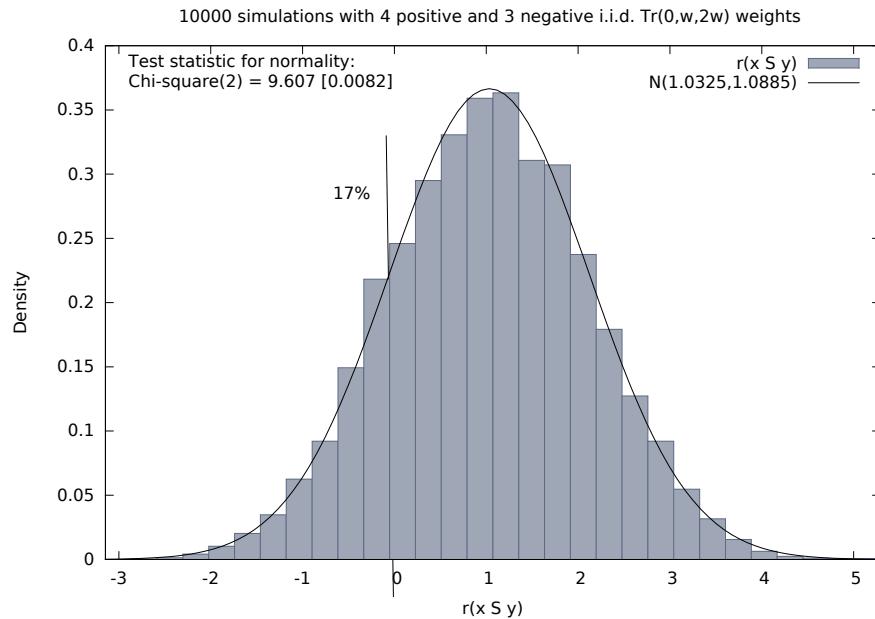


Fig. 18.3 Distribution of 10000 random outranking characteristic values.

18.3 Confidence level of outranking situations

Now, following the classical outranking approach (see [BIS-2013p]), we may say, from an epistemic perspective, that decision action x *outranks* decision action y at *confidence level* $\alpha\%$, when

- An expected majority of criteria validates, at confidence level $\alpha\%$ or higher, a global “*at least as good as*” situation between x and y , and
- no considerably less performing is observed on a discordant criterion.

Dually, decision action x *does not outrank* decision action y at confidence level $\alpha\%$, when

- An expected majority of criteria at confidence level $\alpha\%$ or higher, invalidates a global “*at least as good as*” situation between x and y , and
- no considerably better performing situation is observed on a discordant criterion.

Time for a coded example Let us consider the following random performance tableau.

```

1 >>> from randomPerfTabs import RandomPerformanceTableau
2 >>> t = RandomPerformanceTableau(\n3 ...           numberOfActions=7,\n4 ...           numberOfCriteria=7, seed=100)\n5 >>> t.showPerformanceTableau(Transposed=True)\n6 *----- performance tableau -----*
```

	criteria	weights	'a1'	'a2'	'a3'	'a4'	'a5'	'a6'	'a7'
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 require a confidence level of $\alpha = 90\%$.

The `ConfidentBipolarOutrankingDigraph` class provides such a construction.

```

1 >>> from outrankingDigraphs import \
2 ...             ConfidentBipolarOutrankingDigraph
3 >>> g90 = ConfidentBipolarOutrankingDigraph(t,confidence=90)
4 >>> g90
5     *----- Object instance description -----*
6     Instance class      : ConfidentBipolarOutrankingDigraph
7     Instance name       : rel_randomperftab_CLT
8     Actions            : 7
9     Criteria           : 7
10    Size               : 15
11    Uncertainty model : triangular(a=0,b=2w)
12    Likelihood domain : [-1.0;+1.0]
13    Confidence level   : 0.80 (90.0%)
14    Confident majority : 0.14 (57.1%)
15    Determinateness (%) : 62.07
16    Valuation domain   : [-1.00;1.00]
17    Attributes          : ['name', 'bipolarConfidenceLevel',
18                           'distribution', 'betaParameter', 'actions',
19                           'order', 'valuationdomain', 'criteria',
20                           'evaluation', 'concordanceRelation',
21                           'vetos', 'negativeVetos',
22                           'largePerformanceDifferencesCount',
23                           'likelihoods', 'confidenceCutLevel',
24                           'relation', 'gamma', 'notGamma']

```

The resulting 90%-confident expected outranking relation is shown below.

```

1 >>> g90.showRelationTable(LikelihoodDenotation=True)
2 * ----- Outranking Relation Table -----
3 r / (Ih) | 'a1'   'a2'   'a3'   'a4'   'a5'   'a6'   'a7'
4
5   'a1' | +0.00  +0.71  +0.29  +0.29  +0.29  +0.29  +0.00
6   | (-)  (+1.00)  (+0.95)  (+0.95)  (+0.95)  (+0.95)  (+0.65)
7   'a2' | -0.71  +0.00  -0.29  +0.00  +0.00  +0.29  -0.57
8   | (-1.00)  (-)  (-0.95)  (-0.65)  (+0.73)  (+0.95)  (-1.00)
9   'a3' | -0.29  +0.29  +0.00  -0.29  +0.00  +0.00  -0.29
10  | (-0.95)  (+0.95)  (-)  (-0.95)  (-0.73)  (-0.00)  (-0.95)
11  'a4' | +0.00  +0.00  +0.57  +0.00  +0.29  +0.57  -0.43
12  | (-0.00)  (+0.65)  (+1.00)  (-)  (+0.95)  (+1.00)  (-0.99)
13  'a5' | -0.29  +0.00  +0.00  +0.00  +0.00  +0.29  -0.29
14  | (-0.95)  (-0.00)  (+0.73)  (-0.00)  (-)  (+0.99)  (-0.95)
15  'a6' | -0.29  +0.00  +0.00  -0.29  +0.00  +0.00  +0.00
16  | (-0.95)  (-0.00)  (+0.73)  (-0.95)  (+0.73)  (-)  (-0.00)
17  'a7' | +0.00  +0.71  +0.57  +0.43  +0.29  +0.00  +0.00
18   | (-0.65)  (+1.00)  (+1.00)  (+0.99)  (+0.95)  (-0.00)  (-)

```

```

19     Valuation domain   : [-1.000; +1.000]
20     Uncertainty model : triangular(a=2.0,b=2.0)
21     Likelihood domain : [-1.0;+1.0]
22     Confidence level  : 0.80 (90.0%)
23     Confident majority : 0.14 (57.1%)
24     Determinateness    : 0.24 (62.1%)

```

The (lh) figures, indicated in the table above, correspond to bipolar likelihoods and the required bipolar confidence level equals $(0.90 + 1.0)/2 = 0.80$ (see Line 22 above). Action 'a1' thus confidently outranks all other actions, except 'a7' where the actual likelihood ($+0.65$) is lower than the required one (0.80) and we furthermore observe a considerable counter-performance on criterion 'g1'.

Notice also the lack of confidence in the outranking situations we observe between action 'a2' and actions 'a4' and 'a5'. In the deterministic case we would have $r(a2 \geq a4) = -0.143$ and $r(a2 \geq a5) = +0.143$. All outranking situations with a characteristic value lower 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).

We may draw the corresponding strict 90%-confident outranking digraph, oriented by its initial and terminal strict prekernels (see Fig. 18.4).

```

1 >>> gcd90 = ~ (-g90)
2 >>> gcd90.showPreKernels()
3     *--- Computing preKernels ---*
4     Dominant preKernels :
5         ['a1', 'a7']
6             independence : 0.0
7             dominance   : 0.2857
8             absorbency  : -0.7143
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     ...           bestChoice=['a1', 'a7'], worstChoice=['a2', 'a5',
18     'a6'])
19     *--- exporting a dot file for GraphViz tools -----*
20     Exporting to confidentOutranking.dot
21     dot -Grankdir=BT -Tpng confidentOutranking.dot \
          -o confidentOutranking.png

```

Now, what becomes this 90%-confident outranking digraph when we require a stronger confidence level of, say 99%?

```

1 >>> g99 = ConfidentBipolarOutrankingDigraph(t, confidence=99)
2 >>> g99.showRelationTable()
3     *--- Outranking Relation Table ----
4     r / (lh) | 'a1'   'a2'   'a3'   'a4'   'a5'   'a6'   'a7'
5     -----|-----
6     'a1'   | +0.00  +0.71  +0.00  +0.00  +0.00  +0.00  +0.00
7           | ( - )  (+1.00) (+0.95) (+0.95) (+0.95) (+0.95) (+0.65)
8     'a2'   | -0.71  +0.00  +0.00  +0.00  +0.00  +0.00  -0.57
9           | (-1.00) ( - )  (-0.95) (-0.65) (+0.73) (+0.95) (-1.00)

```

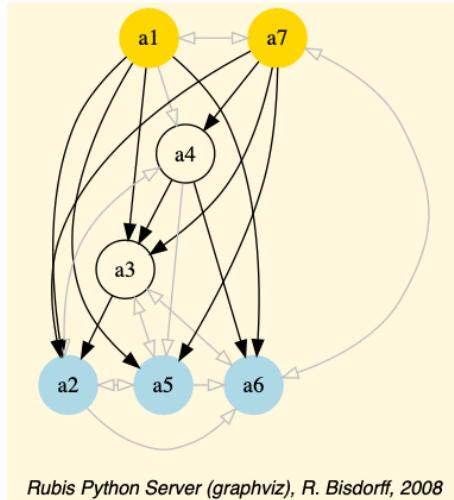


Fig. 18.4 90%-confident strict outranking digraph oriented by its prekernels

Rubis Python Server (graphviz), R. Bisdorff, 2008

10	'a3'	+0.00	+0.00	+0.00	+0.00	+0.00	+0.00	+0.00
11		(-0.95)	(+0.95)	(-)	(-0.95)	(-0.73)	(-0.00)	(-0.95)
12	'a4'	+0.00	+0.00	+0.57	+0.00	+0.00	+0.57	-0.43
13		(-0.00)	(+0.65)	(+1.00)	(-)	(+0.95)	(+1.00)	(-0.99)
14	'a5'	+0.00	+0.00	+0.00	+0.00	+0.00	+0.29	+0.00
15		(-0.95)	(-0.00)	(+0.73)	(-0.00)	(-)	(+0.99)	(-0.95)
16	'a6'	+0.00	+0.00	+0.00	+0.00	+0.00	+0.00	+0.00
17		(-0.95)	(-0.00)	(+0.73)	(-0.95)	(+0.73)	(-)	(-0.00)
18	'a7'	+0.00	+0.71	+0.57	+0.43	+0.00	+0.00	+0.00
19		(-0.65)	(+1.00)	(+1.00)	(+0.99)	(+0.95)	(-0.00)	(-)
20	Valuation domain	:	[-1.000; +1.000]					
21	Uncertainty model	:	triangular(a=2.0,b=2.0)					
22	Likelihood domain	:	[-1.0; +1.0]					
23	Confidence level	:	0.98 (99.0%)					
24	Confident majority	:	0.29 (64.3%)					
25	Determinateness	:	0.13 (56.6%)					

At 99% confidence, the minimal required significance majority support amounts to 64.3% (see Line 24 above). As a result, most outranking situations don't 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 .1% to 56.6% (see Line 25).

Finally, what becomes the previous 90%-confident outranking digraph if the uncertainty concerning the criteria significance weights is modelled with a larger variance, like *uniform* variates (see Line 2 below).

```

1 >>> gu90 = ConfidentBipolarOutrankingDigraph(t,\n2 ...           confidence=90,distribution='uniform')\n3 >>> gu90.showRelationTable()\n4 * ----- Outranking Relation Table -----*\n5 r/(1h) | 'a1'   'a2'   'a3'   'a4'   'a5'   'a6'   'a7'\n6 -----\n7 'a1' | +0.00   +0.71   +0.29   +0.29   +0.29   +0.29   +0.00\n8     | ( - )  (+1.00)  (+0.84)  (+0.84)  (+0.84)  (+0.84)  (+0.49)\n9 'a2' | -0.71   +0.00   -0.29   +0.00   +0.00   +0.29   -0.57\n10    | (-1.00)  ( - )  (-0.84)  (-0.49)  (+0.56)  (+0.84)  (-1.00)\n11 'a3' | -0.29   +0.29   +0.00   -0.29   +0.00   +0.00   -0.29

```

```

12   ' a4 ' | (-0.84) (+0.84) ( - ) (-0.84) (-0.56) (-0.00) (-0.84)
13     | +0.00 +0.00 +0.57 +0.00 +0.29 +0.57 -0.43
14     | (-0.00) (+0.49) (+1.00) ( - ) (+0.84) (+1.00) (-0.95)
15   ' a5 ' | -0.29 +0.00 +0.00 +0.00 +0.00 +0.29 -0.29
16     | (-0.84) (-0.00) (+0.56) (-0.00) ( - ) (+0.92) (-0.84)
17   ' a6 ' | -0.29 +0.00 +0.00 -0.29 +0.00 +0.00 +0.00
18     | (-0.84) (-0.00) (+0.56) (-0.84) (+0.56) ( - ) (-0.00)
19   ' a7 ' | +0.00 +0.71 +0.57 +0.43 +0.29 +0.00 +0.00
20     | (-0.49) (+1.00) (+1.00) (+0.95) (+0.84) (-0.00) ( - )
21 Valuation domain : [-1.000; +1.000]
22 Uncertainty model : uniform(a=2.0,b=2.0)
23 Likelihood domain : [-1.0;+1.0]
24 Confidence level : 0.80 (90.0%)
25 Confident majority : 0.14 (57.1%)
26 Determinateness : 0.24 (62.1%)

```

Despite lower likelihood values (see the *g90* relation table above), we keep the same confident majority level of 57.1% (see Line 25 above) and, hence, also the same 90%-confident outranking digraph.

For concluding, it is worthwhile noticing again that it is in fact the *neutral* value of our bipolar-valued epistemic logic that allows us to easily handle $\alpha\%$ confidence or not of outranking situations when confronted with uncertain criteria significances. 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.

Chapter 19

On robust outrankings with ordinal criteria significances

Abstract

19.1 Cardinal or ordinal criteria significances

The required cardinal significance weights of the performance criteria represent the '*Achilles'* heel of our 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.

A random example of such a decision problem may be generated with the `Random3ObjectivesPerformanceTableau` class.

Listing 19.1 Generate a Random 3 Objectives Performance Tableau

```
1 >>> from randomPerfTabs import \
2 ...           Random3ObjectivesPerformanceTableau
3 >>> t = Random3ObjectivesPerformanceTableau(\
4 ...           numberOfActions=7,\ \
5 ...           numberOfCriteria=9,seed=102)
6 >>> t
7      ----- PerformanceTableau instance description -----
8 Instance class   : Random3ObjectivesPerformanceTableau
9 Seed             : 102
10 Instance name   : random3ObjectivesPerfTab
11 # Actions        : 7
12 # Objectives     : 3
13 # Criteria       : 9
14 Attributes       : ['name', 'valueDigits', 'BigData',
15                      'OrdinalScales', 'missingDataProbability',
16                      'negativeWeightProbability', 'randomSeed',
17                      'sumWeights', 'valuationPrecision',
18                      'commonScale', 'objectiveSupportingTypes',
19                      'actions', 'objectives', 'criteriaWeightMode',
20                      'criteria', 'evaluation', 'weightPreorder']
```

```

21 >>> t.showObjectives()
22 *----- show objectives -----*
23 Eco: Economical aspect
24     ec1 criterion of objective Eco 8
25     ec4 criterion of objective Eco 8
26     ec8 criterion of objective Eco 8
27     Total weight: 24.00 (3 criteria)
28 Soc: Societal aspect
29     so2 criterion of objective Soc 12
30     so7 criterion of objective Soc 12
31     Total weight: 24.00 (2 criteria)
32 Env: Environmental aspect
33     en3 criterion of objective Env 6
34     en5 criterion of objective Env 6
35     en6 criterion of objective Env 6
36     en9 criterion of objective Env 6
37     Total weight: 24.00 (4 criteria)

```

In this example (see Listing 19.1), we face seven decision alternatives that are assessed with respect to three equally important decision objectives concerning: first, an *economical* aspect (Line 24) with a coalition of three performance criteria of significance weight 8, secondly, a *societal* aspect (Line 29) with a coalition of two performance criteria of significance weight 12, and thirdly, an *environmental* aspect (Line 33) with a coalition four performance criteria of significance weight 6.

The question we tackle is the following: How *dependent* on the actual values of the significance weights appears the corresponding bipolar-valued outranking digraph? In the previous Chapter 18, we assumed that the criteria significance weights were random variables. Here, we shall assume that we know for sure only the pre-ordering of the significance weights. In our example we see indeed three increasing weight equivalence classes.

Listing 19.2 The significance weights preorder

```

1 >>> t.showWeightPreorder()
2     ['en3', 'en5', 'en6', 'en9'] (6) <
3     ['ec1', 'ec4', 'ec8'] (8) <
4     ['so2', 'so7'] (12)

```

How stable appear now the outranking situations when assuming only ordinal significance weights?

19.2 Qualifying the stability of outranking situations

Let us construct the normalized bipolar-valued outranking digraph corresponding with the previous 3 Objectives performance tableau t .

Listing 19.3 Example Bipolar Outranking Digraph

```

1 >>> from outrankingDigraphs import BipolarOutrankingDigraph
2 >>> g = BipolarOutrankingDigraph(t)

```

```

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

```

We notice on the principal diagonal the certainly validated reflexive terms +1.00 (see Listing 19.3 Lines 7-13). Now, we know for sure that *unanimous* outranking situations are completely independent of the significance weights. Similarly, all outranking situations that are supported by a majority significance in each coalition of equi-significant criteria are also in fact independent of the actual importance we attach to each individual criteria coalition. But we are also able to test (see [BIS-2014p]) if an outranking situation is independent of all the potential significance weights that respect the given preordering of the weights (see Listing 19.2). Mind that there are, for sure, always outranking situations that are indeed *dependent* on the very values we allocate to the criteria significances.

Such a stability denotation of outranking situations is readily available with the common `showRelationTable()` method.

Listing 19.4 Relation Table with Stability Denotation

```

1 >>> g.showRelationTable(StabilityDenotation=True)
2   * ---- Relation Table ----
3   r/(stab) | 'p1'  'p2'  'p3'  'p4'  'p5'  'p6'  'p7'
4   -----|-----
5   'p1' | +1.00 -0.42 +0.00 -0.69 +0.39 +0.11 -0.06
6   | (+4) (-2) (+0) (-3) (+2) (+2) (-1)
7   'p2' | +0.58 +1.00 +0.83 0.00 +0.58 +0.58 +0.58
8   | (+2) (+4) (+3) (+2) (+2) (+2) (+2)
9   'p3' | +0.25 -0.33 +1.00 0.00 +0.50 +1.00 +0.25
10  | (+2) (-2) (+4) (0) (+2) (+2) (+1)
11  'p4' | +0.78 0.00 +0.61 +1.00 +1.00 +1.00 +0.67
12  | (+3) (-1) (+3) (+4) (+4) (+4) (+2)
13  'p5' | -0.11 -0.50 -0.25 -0.89 +1.00 +0.11 -0.14
14  | (-2) (-2) (-2) (-3) (+4) (+2) (-2)
15  'p6' | +0.22 -0.42 0.00 -1.00 +0.17 +1.00 -0.11
16  | (+2) (-2) (+1) (-2) (+2) (+4) (-2)
17  'p7' | +0.22 -0.50 +0.17 -0.06 +0.78 +0.42 +1.00
18  | (+2) (-2) (+1) (-1) (+3) (+2) (+4)

```

We may thus distinguish the following bipolar-valued stability levels:

- **+4|−4** : *unanimous* outranking, resp. outranked, situation. The pairwise trivial reflexive outrankings, for instance, all show this stability level;
- **+3|−3** : validated outranking, resp. outranked, situation in *each* coalition of equisignificant criteria. This is, for instance, the case for the outranking situation observed between alternatives 'p1' and 'p4' (see Listing 19.4 Lines 6 and 12);

- **+2|−2** : outranking, resp. outranked, situation validated with *all* potential significance weights that are compatible with the given significance preorder (see Listing 19.2). This is case for the comparison of alternatives 'p1' and 'p2' (see Lines 6 and 8);
- **+1|−1** : validated outranking, resp. outranked, situation with the given significance weights, a situation we may observe between alternatives 'p3' and 'p7' (see Lines 10 and 16);
- **0** : indeterminate relational 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, i.e. there is given a single equivalence class containing all the performance criteria, we may only distinguish stability levels ± 4 and ± 3 . Furthermore, when in such a case an outranking (resp. outranked) situation is validated at level +3 (resp. −3) no potential preordering of the criteria significances exists that could qualify the same situation as outranked (resp. outranking) at level −2 (resp. +2).

In the other limit case, when all performance criteria admit different significances, i.e. the significance weights may be linearly ordered, no stability level +3 or −3 may be observed.

As mentioned above, all reflexive comparisons confirm an unanimous outranking situation: all decision alternatives are indeed trivially “*as well performing 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 Lines 14 and 16).

Let us inspect the details of how alternatives 'p4' and 'p5' compare.

```

1 >>> g.showPairwiseComparison('p4','p5')
2 *----- pairwise comparison -----*
3 Comparing actions : (p4, p5)
4 crit. wght.   g(x)   g(y)    diff | ind   pref   r()
5 ec1    8.00  85.19  46.75  +38.44 | 5.00  10.00  +8.00
6 ec4    8.00  72.26   8.96  +63.30 | 5.00  10.00  +8.00
7 ec8    8.00  44.62  35.91   +8.71 | 5.00  10.00  +8.00
8 en3    6.00  80.81  31.05  +49.76 | 5.00  10.00  +6.00
9 en5    6.00  49.69  29.52  +20.17 | 5.00  10.00  +6.00
10 en6   6.00  66.21  31.22  +34.99 | 5.00  10.00  +6.00
11 en9   6.00  50.92   9.83  +41.09 | 5.00  10.00  +6.00
12 so2  12.00  49.05  12.36  +36.69 | 5.00  10.00  +12.00
13 so7  12.00  55.57  44.92  +10.65 | 5.00  10.00  +12.00
14 Valuation in range: -72.00 to +72.00; -----
15                                     global concordance: +72.00

```

Alternative $p4$ is indeed performing unanimously “*at least as well as*” alternative ' $p5$ ' and $r(p4 \succsim p5) = 72/72 = +1.00$ (see Listing 19.4 Line 11).

The converse comparison does not, however, deliver such an unanimous outranked situation.

```

1 >>> g.showPairwiseComparison('p5','p4')
2 *----- pairwise comparison -----*

```

```

3 Comparing actions : (p5, p4)
4 crit. wght. g(x) g(y) diff | ind pref r()
5 ec1 8.00 46.75 85.19 -38.44 | 5.00 10.00 -8.00
6 ec4 8.00 8.96 72.26 -63.30 | 5.00 10.00 -8.00
7 ec8 8.00 35.91 44.62 -8.71 | 5.00 10.00 +0.00
8 en3 6.00 31.05 80.81 -49.76 | 5.00 10.00 -6.00
9 en5 6.00 29.52 49.69 -20.17 | 5.00 10.00 -6.00
10 en6 6.00 31.22 66.21 -34.99 | 5.00 10.00 -6.00
11 en9 6.00 9.83 50.92 -41.09 | 5.00 10.00 -6.00
12 so2 12.00 12.36 49.05 -36.69 | 5.00 10.00 -12.00
13 so7 12.00 44.92 55.57 -10.65 | 5.00 10.00 -12.00
14 Valuation in range: -72.00 to +72.00; -----
15 global concordance: -64.00

```

This comparison only qualifies at stability level -3 (see Listing 19.4 Line 13); $r(p5 \succsim p4) = -64/72 = -0.89$. Indeed, on criterion ec8 we observe a small negative performance difference of -8.71 (see Line 7 above) which is effectively below the supposed preference discrimination threshold of 10.00. Yet, the outranked situation is supported by a majority of criteria in each decision objective. Hence, the reported preferential situation is completely independent of any chosen significance weights.

Let us now consider a comparison, like the one between alternatives 'p2' and 'p1', that is qualified at stability level +2, resp. -2.

Listing 19.5 Comparison of alternatives 'p2' and 'p1'

```

1 >>> g.showPairwiseOutrankings('p2','p1')
2 *----- pairwise comparison -----*
3 Comparing actions : (p2, p1)
4 crit. wght. g(x) g(y) diff | ind pref r()
5 ec1 8.00 89.77 38.11 +51.66 | 5.00 10.00 +8.00
6 ec4 8.00 86.00 22.65 +63.35 | 5.00 10.00 +8.00
7 ec8 8.00 89.43 77.02 +12.41 | 5.00 10.00 +8.00
8 en3 6.00 20.79 58.16 -37.37 | 5.00 10.00 -6.00
9 en5 6.00 23.83 31.40 -7.57 | 5.00 10.00 +0.00
10 en6 6.00 18.66 11.41 +7.25 | 5.00 10.00 +6.00
11 en9 6.00 26.65 44.37 -17.72 | 5.00 10.00 -6.00
12 so2 12.00 89.12 22.43 +66.69 | 5.00 10.00 +12.00
13 so7 12.00 84.73 28.41 +56.32 | 5.00 10.00 +12.00
14 Valuation in range: -72.00 to +72.00; -----
15 global concordance: +42.00
16 *----- pairwise comparison -----*
17 Comparing actions : ('p1', 'p2')
18 crit. wght. g(x) g(y) diff | ind pref r()
19 ec1 8.00 38.11 89.77 -51.66 | 5.00 10.00 -8.00
20 ec4 8.00 22.65 86.00 -63.35 | 5.00 10.00 -8.00
21 ec8 8.00 77.02 89.43 -12.41 | 5.00 10.00 -8.00
22 en3 6.00 58.16 20.79 +37.37 | 5.00 10.00 +6.00
23 en5 6.00 31.40 23.83 +7.57 | 5.00 10.00 +6.00
24 en6 6.00 11.41 18.66 -7.25 | 5.00 10.00 +0.00
25 en9 6.00 44.37 26.65 +17.72 | 5.00 10.00 +6.00
26 so2 12.00 22.43 89.12 -66.69 | 5.00 10.00 -12.00
27 so7 12.00 28.41 84.73 -56.32 | 5.00 10.00 -12.00
28 Valuation in range: -72.00 to +72.00; -----
29 global concordance: -30.00

```

In both comparisons, the performances observed with respect to the environmental decision objective are not validating with a significant majority the otherwise unanimous outranking, resp. outranked situations. 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).

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')
2 *----- pairwise comparison -----
3 Comparing actions : ('p7', 'p3')
4 crit. wght. g(x) g(y) diff | ind pref r()
5 ecl 8.00 15.33 80.19 -64.86 | 5.00 10.00 -8.00
6 ec4 8.00 36.31 68.70 -32.39 | 5.00 10.00 -8.00
7 ec8 8.00 38.31 91.94 -53.63 | 5.00 10.00 -8.00
8 en3 6.00 30.70 46.78 -16.08 | 5.00 10.00 -6.00
9 en5 6.00 35.52 27.25 +8.27 | 5.00 10.00 +6.00
10 en6 6.00 69.71 1.65 +68.06 | 5.00 10.00 +6.00
11 en9 6.00 13.10 14.85 -1.75 | 5.00 10.00 +6.00
12 so2 12.00 68.06 58.85 +9.21 | 5.00 10.00 +12.00
13 so7 12.00 58.45 15.49 +42.96 | 5.00 10.00 +12.00
14 Valuation in range: -72.00 to +72.00; -----
15 global concordance: +12.00
16 *----- pairwise comparison -----
17 Comparing actions : ('p3', 'p7')
18 crit. wght. g(x) g(y) diff | ind pref r()
19 ecl 8.00 80.19 15.33 +64.86 | 5.00 10.00 +8.00
20 ec4 8.00 68.70 36.31 +32.39 | 5.00 10.00 +8.00
21 ec8 8.00 91.94 38.31 +53.63 | 5.00 10.00 +8.00
22 en3 6.00 46.78 30.70 +16.08 | 5.00 10.00 +6.00
23 en5 6.00 27.25 35.52 -8.27 | 5.00 10.00 +0.00
24 en6 6.00 1.65 69.71 -68.06 | 5.00 10.00 -6.00
25 en9 6.00 14.85 13.10 +1.75 | 5.00 10.00 +6.00
26 so2 12.00 58.85 68.06 -9.21 | 5.00 10.00 +0.00
27 so7 12.00 15.49 58.45 -42.96 | 5.00 10.00 -12.00
28 Valuation in range: -72.00 to +72.00; -----
29 global concordance: +18.00

```

In both cases, choosing significances that are just 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 (see [BIS-2004-1p], [BIS-2004-2p]).

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 increasing order (see Listing 19.2). 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 $q[i]$ 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 'p2' and 'p1', for instance (see Listing 19.5), we observe the following counts:

$W[i]$	-12	-8	-6	0	6	8	12
$q[i]$	0	0	2	1	1	3	2
$Q[i]$	0	0	2	3	4	7	9

Let us denote $-q$ and $-Q$ the reversed versions of the q and the Q lists. We thus obtain the following result.

$W[i]$	-12	-8	-6	0	6	8	12
$-q[i]$	2	3	1	1	2	0	0
$-Q[i]$	2	5	6	7	9	9	9

Now, a pairwise outranking situation will be qualified at stability level +2, i.e. positively validated with any significance weights that are compatible with the given weights preorder, when for all i , we observe $Q[i] \leq -Q[i]$ and there exists one i such that $Q[i] < -Q[i]$. Similarly, a pairwise outranked situation will be qualified at stability level -2, when for all i , we observe $Q[i] \geq -Q[i]$ and there exists one i such that $Q[i] > -Q[i]$ (see [BIS-2004-2p]).

We may verify, for instance, that the outranking situation observed between alternatives 'p2' and 'p1' does indeed verify this *first order distributional dominance* condition.

$W[i]$	-12	-8	-6	0	6	8	12
$Q[i]$	0	0	2	3	4	7	9
$-Q[i]$	2	5	6	7	9	9	9

Notice that outranking situations qualified at stability levels 4 and 3, evidently also verify the stability level 2 test above. The outranking situation between alternatives 'p7' and 'p3' does not, however, verify this test (see Listing 19.6).

$W[i]$	-12	-8	-6	0	6	8	12
$q[i]$	0	3	1	0	3	0	2
$Q[i]$	0	3	4	4	7	7	9
$-Q[i]$	2	2	5	5	6	9	9

This time, not all the $Q[i]$ are *lower or equal* than the corresponding $-Q[i]$ terms. Hence the outranking situation between 'p7' and 'p3' is not positively validated with all potential significance weights that are compatible with the given weights preorder.

Using this stability denotation, we may, hence, define the following *robust* version of a bipolar-valued outranking digraph.

19.4 Robust bipolar-valued outranking digraphs

We say that decision alternative x *robustly outranks* decision alternative y when:

- x positively outranks y at stability level 2 or higher and
- we may not observe any considerable counter-performance of x on a discordant criterion.

Dually, we say that decision alternative x *does not robustly outrank* decision alternative y when:

- x is positively outranked by y at stability level 2 or higher and
- we may not observe any considerable better performance of x on a discordant criterion.

The corresponding *robust* outranking digraph may be computed with the `RobustOutrankingDigraph` class as follows:

Listing 19.7 Computing a robust outranking digraph

```

1 >>> from outrankingDigraphs import\
2 ...           RobustOutrankingDigraph
3 >>> rg = RobustOutrankingDigraph(t) # same t as before
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(StabilityDenotation=True)
20 * ---- Relation Table ----
21 r/(stab) | 'p1'  'p2'  'p3'  'p4'  'p5'  'p6'  'p7'
22 -----|-----
23 'p1'    | +1.00 -0.42 +0.00 -0.69 +0.39 +0.11 +0.00
24           | (+4)  (-2)  (+0)  (-3)  (+2)  (+2)  (-1)
```

25	'p2'	+0.58 +1.00 +0.83 +0.00 +0.58 +0.58 +0.58
26		(+2) (+4) (+3) (+2) (+2) (+2) (+2)
27	'p3'	+0.25 -0.33 +1.00 +0.00 +0.50 +1.00 +0.00
28		(+2) (-2) (+4) (+0) (+2) (+2) (+1)
29	'p4'	+0.78 +0.00 +0.61 +1.00 +1.00 +1.00 +0.67
30		(+3) (-1) (+3) (+4) (+4) (+4) (+2)
31	'p5'	-0.11 -0.50 -0.25 -0.89 +1.00 +0.11 -0.14
32		(-2) (-2) (-2) (-3) (+4) (+2) (-2)
33	'p6'	+0.22 -0.42 +0.00 -1.00 +0.17 +1.00 -0.11
34		(+2) (-2) (+1) (-2) (+2) (+4) (-2)
35	'p7'	+0.22 -0.50 +0.00 +0.00 +0.78 +0.42 +1.00
36		(+2) (-2) (+1) (-1) (+3) (+2) (+4)

We may notice that all outranking situations, qualified at stability level ± 1 , are now put to an *indeterminate* status. In the example here, we actually drop three positive outrankings: between 'p3' and 'p7', between 'p7' and 'p3', and between 'p6' and 'p3', where the last situation is actually already put to doubt by a veto situation (see Listing 19.7 Lines 22-35). We drop as well three negative outrankings: between 'p1' and 'p7', between 'p4' and 'p2', and between 'p7' and 'p4' (see Lines 22-35).

Notice by the way that outranking or outranked situations, although qualified at level ± 2 or ± 3 may nevertheless be put to doubt by considerable performance differences. We may observe such an outranking situation when comparing, for instance, alternatives 'p2' and 'p4' (see Listing 19.7 Lines 24-25).

Listing 19.8 Comparing alternatives 'p2' and 'p4'

1	>>> rg.showPairwiseComparison('p2','p4')
2	*----- pairwise comparison -----*
3	Comparing actions : (p2, p4)
4	crit. wght. g(x) g(y) diff ind pref r() v veto
5	-----
6	ec1 8.00 89.77 85.19 +4.58 5.00 10.00 +8.00
7	ec4 8.00 86.00 72.26 +13.74 5.00 10.00 +8.00
8	ec8 8.00 89.43 44.62 +44.81 5.00 10.00 +8.00
9	en3 6.00 20.79 80.81 -60.02 5.00 10.00 -6.00 60.00 -1.00
10	en5 6.00 23.83 49.69 -25.86 5.00 10.00 -6.00
11	en6 6.00 18.66 66.21 -47.55 5.00 10.00 -6.00
12	en9 6.00 26.65 50.92 -24.27 5.00 10.00 -6.00
13	so2 12.00 89.12 49.05 +40.07 5.00 10.00 +12.00
14	so7 12.00 84.73 55.57 +29.16 5.00 10.00 +12.00
15	Valuation in range: -72.00 to +72.00; -----
16	global concordance: +24.00

Despite being robust, the apparent positive outranking situation between alternatives 'p2' and 'p4' is indeed put to doubt by a considerable counter-performance (-60.02) of 'p2' observed on criterion 'en3', a negative difference which exceeds slightly the assumed veto discrimination threshold $v = 60.00$ (see Listing 19.8 Line 9).

We may finally compare in Fig. 19.1 the *standard* and the *robust* version of the corresponding strict outranking digraphs, both oriented by their respective identical initial and terminal prekernels.

The robust version drops two strict outranking situations: between 'p4' and 'p7' and between 'p7' and 'p1'. The remaining 14 strict outranking (resp. outranked) situations are now all verified at a stability level of $+2$ and more (resp. -2 and less).

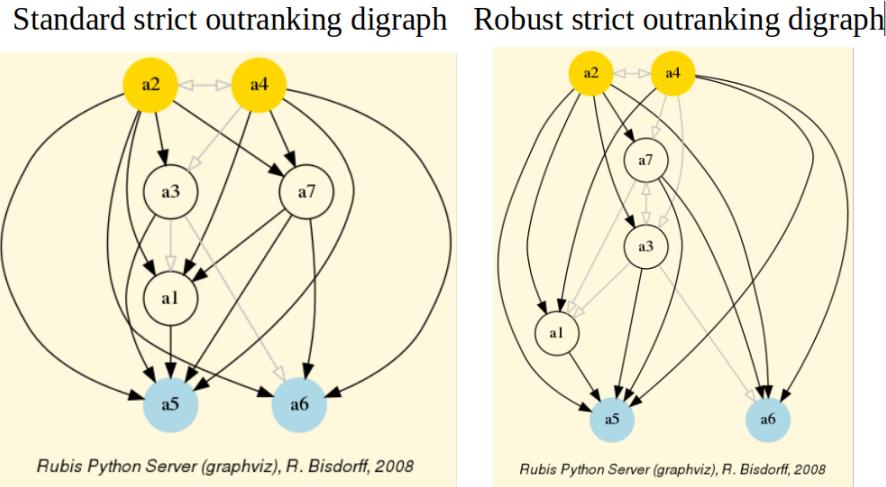


Fig. 19.1 Standard versus robust strict outranking digraphs oriented by their initial and terminal prekernels

They are, hence, only depending on potential significance weights that must respect the given significance preorder (see Listing 19.2).

Heatmap of Performance Tableau '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	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: **standard**, Ranking rule: **NetFlows**

Ordinal (Kendall) correlation between global ranking and global outranking relation: **+0.942**

Mean marginal correlation (a) : **+0.338**

Fig. 19.2 Heat map of the random 3 objectives performance tableau ordered by the NETFLOWS ranking rule.

To appreciate the apparent orientation of the standard and robust strict outranking digraphs shown in Fig. 19.1, let us have a final heat map view in Fig. 19.2 on the underlying performance tableau ordered by the NETFLOWS ranking rule.

```
1 >>> t.showHTMLPerformanceHeatmap(\n2 ...     Correlations=True, rankingRule='NetFlows')
```

As the initial prekernel is here validated at stability level +2, recommending alternatives 'p4', as well as 'p2', as potential best choices, appears well justified. Alternative 'p4' represents indeed an overall *best compromise choice* between all decision objectives, whereas alternative 'p2' gives an unanimous best choice with respect to two out of three decision objectives. Up to the decision maker to make his final choice.

When facing a performance tableau involving multiple decision objectives, the robustness level ± 3 may lead to distinguishing what we call *unopposed* outranking situations, like the one shown between alternative 'p4' and 'p1' ($r(p4 \succsim p1) = +0.78$, see Listing 19.4 Line 11), namely preferential situations that are more or less validated or invalidated by all the decision objectives.

19.5 Characterising unopposed multiobjective outranking situations

Formally, we say that decision alternative x *outranks* decision alternative y *unopposed* when:

x positively outranks y on one or more decision objective without x being positively outranked by y on any decision objective.

Dually, we say that decision alternative x *does not outrank* decision alternative y *unopposed* when:

x is positively outranked by y on one or more decision objective without x positively outranking y on any decision objective.

Let us reconsider, for instance, the performance tableau with three decision objectives already seen in Listing 19.1:

```
1 >>> from randomPerfTabs import \
2 ...     Random3ObjectivesPerformanceTableau
3 >>> t = Random3ObjectivesPerformanceTableau(\
4 ...     numberOfActions=7, \
5 ...     numberOfCriteria=9, seed=102)
6 >>> t.showObjectives()
7     ----- show objectives -----
8     Eco: Economical aspect
9         ec1 criterion of objective Eco 8
10        ec4 criterion of objective Eco 8
11        ec8 criterion of objective Eco 8
12     Total weight: 24.00 (3 criteria)
```

```

13     Soc: Societal aspect
14         so2 criterion of objective Soc 12
15         so7 criterion of objective Soc 12
16     Total weight: 24.00 (2 criteria)
17     Env: Environmental aspect
18         en3 criterion of objective Env 6
19         en5 criterion of objective Env 6
20         en6 criterion of objective Env 6
21         en9 criterion of objective Env 6
22     Total weight: 24.00 (4 criteria)

```

We notice in this example three decision objectives of equal importance (see Lines 8,13,17). What will be the outranking situations that are positively (resp. negatively) validated for each one of the decision objectives taken individually ?

We may obtain such unopposed multiobjective outranking situations by operating an epistemic *average fusion* (see the `symmetricAverage()` method) of the marginal outranking digraphs restricted to the coalition of criteria supporting each one of the decision objectives (see Listing 19.9 below).

Listing 19.9 Computing unopposed outranking situations

```

1 >>> from outrankingDigraphs import\
2 ...             BipolarOutrankingDigraph
3 >>> geco = BipolarOutrankingDigraph(t,\n4 ...                         objectivesSubset=['Eco'])
5 >>> gsoc = BipolarOutrankingDigraph(t,\n6 ...                         objectivesSubset=['Soc'])
7 >>> genv = BipolarOutrankingDigraph(t,\n8 ...                         objectivesSubset=['Env'])
9 >>> from digraphs import FusionLDigraph
10 >>> objectiveWeights = \
11 ...             [t.objectives[obj]['weight']\n12 ...             for obj in t.objectives]
13 >>> uopg = FusionLDigraph([geco,gsoc,genv],\
14 ...                         operator='o-average',\
15 ...                         weights=objectiveWeights)
16 >>> uopg.showRelationTable(ReflexiveTerms=False)
17 * ---- Relation Table ----
18 r | 'p1'   'p2'   'p3'   'p4'   'p5'   'p6'   'p7'
19 -----|-----
20 'p1' |   -    +0.00  +0.00  -0.69  +0.39  +0.11  +0.00
21 'p2' |  +0.00   -    +0.83  +0.00  +0.00  +0.00  +0.00
22 'p3' |  +0.00  -0.33   -    +0.00  +0.50  +0.00  +0.00
23 'p4' |  +0.78  +0.00  +0.61   -    +1.00  +1.00  +0.67
24 'p5' |  -0.11  +0.00  +0.00  -0.89   -    +0.11  +0.00
25 'p6' |  +0.00  +0.00  +0.00  -0.44  +0.17   -    +0.00
26 'p7' |  +0.00  +0.00  +0.00  +0.00  +0.78  +0.42   -
27 Valuation domain: [-1.0; 1.0]

```

Positive (resp. negative) $r(x \succsim y)$ characteristic values, like $r(p1 \succsim p5) = +0.39$ (see Listing 19.9 Line 20), show hence only outranking situations being validated (resp. invalidated) by one or more decision objectives without being invalidated (resp. 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 19.10 Unopposed outranking digraph constructor

```
1 >>> from outrankingDigraphs import\_
2 ...                      UnOpposedBipolarOutrankingDigraph
3 >>> uopg = UnOpposedBipolarOutrankingDigraph(t)
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', 'marginalRelationsRelations',
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 19.10 Lines 12-13) 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 may now, for instance, verify the unopposed status of the outranking situation observed between alternatives 'p1' and 'p5'.

Listing 19.11 Example of unopposed multiobjective outranking situation

```

1 >>> uopg.showPairwiseComparison('p1','p5')
2 *----- pairwise comparison -----*
3 Comparing actions : ('p1', 'p5')
4 crit. wght. g(x) g(y) diff | ind pref r()
5 ec1 8.00 38.11 46.75 -8.64 | 5.00 10.00 +0.00
6 ec4 8.00 22.65 8.96 +13.69 | 5.00 10.00 +8.00
7 ec8 8.00 77.02 35.91 +41.11 | 5.00 10.00 +8.00
8 en3 6.00 58.16 31.05 +27.11 | 5.00 10.00 +6.00
9 en5 6.00 31.40 29.52 +1.88 | 5.00 10.00 +6.00
10 en6 6.00 11.41 31.22 -19.81 | 5.00 10.00 -6.00
11 en9 6.00 44.37 9.83 +34.54 | 5.00 10.00 +6.00
12 so2 12.00 22.43 12.36 +10.07 | 5.00 10.00 +12.00
13 so7 12.00 28.41 44.92 -16.51 | 5.00 10.00 -12.00
14 Valuation in range: -72.00 to +72.00;
15 global concordance: +28.00

```

In Listing 19.11 we see that alternative 'p1' does indeed positively outrank alternative 'p5' from the economic perspective ($r(p1 \succsim_{Eco} p5) = +16/24$) as well as from the environmental perspective ($r(p1 \succsim_{Env} p5) = +12/24$). Whereas, from the societal perspective, both alternatives appear incomparable ($r(p1 \succsim_{Soc} p5) = 0/24$).

When fixed proportional criteria significances 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

Indeed, best choice recommendations, computed from an *unopposed multiobjective* outranking digraph, will in fact deliver *Pareto efficient* choices.

Listing 19.12 Pareto efficient multiobjective choice

```

1 >>> uopg.showBestChoiceRecommendation()
2 Best choice recommendation(s) (BCR)
3   (in decreasing order of determinateness)
4 Credibility domain: [-1.00,1.00]
5 === >> potential best choice(s)
6 choice          : ['p2', 'p4', 'p7']
7   independence   : 0.00
8   dominance      : 0.33
9   absorbency     : 0.00
10  covering (%)   : 33.33
11  determinateness (%) : 50.00
12 === >> potential worst choice(s)
13 choice          : ['p3', 'p5', 'p6', 'p7']
14   independence   : 0.00
15   dominance      : -0.61
16   absorbency     : 0.11
17   covered (%)    : 33.33
18   determinateness (%) : 50.00

```

Our previous *robust* best choice recommendation ('p2' and 'p4', see Fig. 19.1) remains, in this example here, *stable*. We recover indeed the best choice recommendation ['p2', 'p4', 'p7'] (see Listing 19.12 Line 6). Yet, notice that decision alternative 'p7' appears to be at the same time a potential *best* as well as a potential *worst* 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.

We may visualize this kind of Pareto efficient result in Fig. 19.3 below.

```

1 >>> (~(-uopg)).exportGraphViz(fileName = 'unopDigraph', \
2 ...                               bestChoice = ['p2','p4'], \
3 ...                               worstChoice = ['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

```

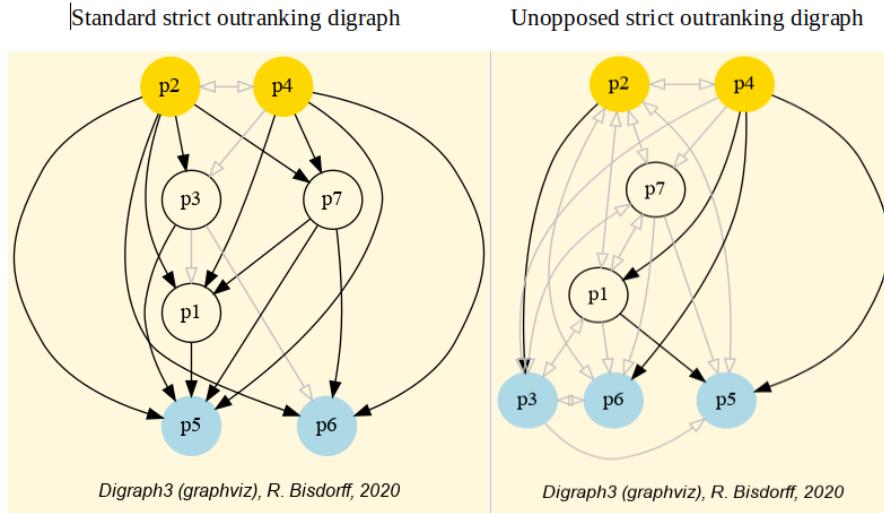


Fig. 19.3 Standard versus *unopposed* strict outranking digraphs oriented by best and worst choice recommendations.

In order to make now an eventual best unique choice, a decision maker will necessarily have to weight, 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 outranking digraph when facing performance tableaux with criteria of only ordinal-valued significances. A real world application of our stability analysis with such a kind of performance tableau may be consulted in [BIS-2015p].

Chapter 20

Two-stage elections with multipartisan primary selection

Abstract

In a *social choice* context, where decision objectives would match different political parties, Pareto efficient choice recommendations represent in fact *multipartisan* social choices that could judiciously deliver the primary selection in a two stage election system.

To compute such Pareto efficient social choices we need to, first, convert a given linear voting profile (with polls) into a corresponding performance tableau.

20.1 Converting voting profiles into performance tableaux

We shall illustrate this point with a voting profile we discussed already in Chapter ??.

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   a06 : 19.91% | a11 : 22.94% | a06 : 15.00%
28   a07 : 14.27% | a08 : 15.65% | a11 : 13.08%
29   a03 : 10.02% | a04 : 15.07% | a08 : 09.01%
30   a13 : 08.39% | a06 : 13.40% | a07 : 08.79%
31   a15 : 08.39% | a03 : 06.49% | a03 : 07.44%
32   a11 : 06.70% | a09 : 05.63% | a04 : 07.11%
33   a01 : 06.17% | a07 : 05.10% | a01 : 05.06%
34   a12 : 04.81% | a01 : 05.09% | a13 : 05.04%
35   a08 : 04.75% | a12 : 03.43% | a15 : 04.23%
36   a10 : 04.66% | a13 : 02.71% | a12 : 03.71%
37   a14 : 04.42% | a14 : 02.70% | a14 : 03.21%
38   a05 : 04.01% | a15 : 00.86% | a09 : 03.10%
39   a09 : 01.40% | a10 : 00.44% | a10 : 02.34%
40   a04 : 01.18% | a05 : 00.29% | a05 : 01.97%
41   a02 : 00.90% | a02 : 00.21% | a02 : 00.51%

```

In this example (see Listing 20.1 Lines 18-), we obtained 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%, appeared to be 'a06' (19.91%), 'a07' (14.27%) and 'a03' (10.02%). Whereas for Party-2 supporters, favorite candidates appeared to be 'a11' (22.94%), followed by 'a08' (15.65%), 'a04' (15.07%) and 'a06' (13.4%).

We may convert this linear voting profile into a PerformanceTableau object where each political party matches to a decision objective.

Listing 20.2 Converting a voting profile into a performance tableau

```

1 >>> lvp.save2PerfTab('votingPerfTab')
2 >>> from perfTabs import PerformanceTableau
3 >>> vpt = PerformanceTableau('votingPerfTab')
4 >>> vpt
5   *----- PerformanceTableau instance description -----
6   Instance class    : PerformanceTableau
7   Instance name     : votingPerfTab
8   Actions           : 15
9   Objectives        : 3
10  Criteria          : 1000
11  Attributes        : ['name', 'actions', 'objectives',
12    'criteria', 'weightPreorder', 'evaluation']
13 >>> vpt.objectives
14  OrderedDict([
15    ('party0', {'name': 'other', 'weight': Decimal('104'),
16      'criteria': ['v0003', 'v0008', 'v0011', ... ]}),
17    ('party1', {'name': 'party 1', 'weight': Decimal('460'),
18      'criteria': ['v0002', 'v0006', 'v0007', ... ]}),
19    ('party2', {'name': 'party 2', 'weight': Decimal('436'),
20      'criteria': ['v0001', 'v0004', 'v0005', ... ]})}

```

21])

In Listing 20.2 we first store the linear voting in a `PerformanceTableau` format (see Line 1). In Line 3, we reload this performance tableau data. The three parties of the linear voting profile represent three decision objectives and the 1000 voters are distributed as 10000 performance criteria according to the party they support.

20.2 Multipartisan primary selection of eligible candidates

In order to operate now a *primary multipartisan selection* of potential election winners, we compute the corresponding unopposed multiobjective outranking digraph (see Section 19.5).

Listing 20.3 Computing unopposed multiobjective outranking situations

```

1 >>> from outrankingDigraphs import \
2 ...     UnOpposedBipolarOutrankingDigraph
3 >>> uog = UnOpposedBipolarOutrankingDigraph(vpt)
4 >>> uog
5      *----- Object instance description -----*
6      Instance class      : UnOpposedBipolarOutrankingDigraph
7      Instance name       : unopposed_outrankings
8      Actions             : 15
9      Criteria            : 1000
10     Size                : 34
11     Oppositeness (%)   : 67.31
12     Determinateness (%) : 57.61
13     Valuation domain   : [-1.00;1.00]
14     Attributes          : ['name', 'actions', 'valuationdomain',
15                           'objectives', 'criteria',
16                           'methodData',
17                           'evaluation', 'order', 'runTimes',
18                           'relation',
                           'marginalRelationsRelations',
                           'gamma', 'notGamma']
```

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%.

We may visualize the corresponding bipolar-valued relation table by orienting the list of candidates with the help of the initial and terminal prekernels.

Listing 20.4 Computing unopposed multiobjective outranking situations

```

1 >>> uog.showPreKernels()
2      *--- Computing preKernels ---*
3      Dominant preKernels :
4      ['a11', 'a06', 'a13', 'a15']
5      independence : 0.0
6      dominance    : 0.18
7      absorbency    : -0.66
```

```

8     covering      : 0.43
9     Absorbent preKernels :
10    ['a02', 'a04', 'a14', 'a03']
11    independence : 0.0
12    dominance   : 0.0
13    absorbency   : 0.37
14    covered      : 0.46
15 >>> orientedCandidatesList = ['a06','a11','a13','a15',\
16 ...          'a01','a05','a07','a08','a09','a10','a12',\
17 ...          'a02','a03','a04','a14']
18 >>> uog.showHTMLRelationTable(\
19 ...      actionsList=orientedCandidatesList,\
20 ...      tableTitle='Unopposed three-partisan outrankings')

```

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	0.55	0.00	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

In Fig. 20.1, we may notice that the dominating outranking prekernel ['a06', 'a11', 'a13', 'a15'] gathers in fact a multipartisan selection of potential election winners. It is worthwhile noticing that in the majority margins obtained from a linear voting profile do verify the zero-sum rule: $(r(x \succsim y) + r(y \succsim x)) = 0.0$. To each positive outranking situation corresponds indeed an equivalent negative converse situation and the resulting outranking and strict outranking digraphs are the same.

20.3 Secondary election winner determination

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,\n3 ..          actionsSubset=['a06','a11','a13','a15'])
4 >>> g2.showRelationTable(ReflexiveTerms=False)
5   * ----- Relation Table -----
6   r    | 'a06'  'a11'  'a13'  'a15'
7   -----|-----
8   'a06' |   -    +0.10  +0.48  +0.52
9   'a11' | -0.10   -    +0.27  +0.29
10  'a13' | -0.48 -0.27   -    +0.19
11  'a15' | -0.52 -0.29 -0.19   -
12  Valuation domain: [-1.0; 1.0]
13 >>> g2.computeCondorcetWinners()
14  ['a06']
15 >>> g2.computeCopelandRanking()
16  ['a06', 'a11', 'a13', 'a15']

```

Candidate 'a06' appears clearly to be the winner of this election. Notice by the way that the restricted pairwise outranking relation shown in Listing 20.5 represents a linear ordering of the preselected candidates.

We may eventually check the quality of this best choice by noticing that candidate 'a06' represents indeed the *simple majority*, the *instant-run-off*, the BORDA, as well as the CONDORCET winner of the initially given linear voting profile *lvp*.

```

1 >>> lvp.computeSimpleMajorityWinner()
2  ['a06']
3 >>> lvp.computeInstantRunoffWinner()
4  ['a06']
5 >>> lvp.computeBordaWinners()
6  ['a06']
7 >>> from votingProfiles import MajorityMarginsDigraph
8 >>> cd = MajorityMarginsDigraph(lvp)
9 >>> cd.computeCondorcetWinners()
10 ['a06']

```

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 btw rejecting the actual winning candidate.

20.4 Multipartisan preferences in divisive politics

However, in a very *divisive* two major parties system, like in the US, where preferences of the supporters of one major party appear to be very opposite to the pref-

erences of the supporters of the other major party, the multipartisan outranking digraph will become nearly indeterminate.

In Listing 20.6 below we generate such a divisive kind of linear voting profile with the help of the `DivisivePolitics` flag Footnote[5] (see Lines 4 and 13-19). When now converting the voting profile into a performance tableau (Lines 20-21), we may compute the corresponding unopposed outranking digraph.

Listing 20.6 A divisive two-party example of a random linear voting profile

```

1 >>> from votingProfiles import RandomLinearVotingProfile

2 >>> lvp = RandomLinearVotingProfile(\ 
3 ...     numberOfCandidates=7,numberOfVoters=500,\ 
4 ...     WithPolls=True, partyRepartition=0.4,other=0.2,\ 
5 ...     DivisivePolitics=True, seed=1)
6 >>> lvp.showRandomPolls()
7     Random repartition of voters
8     Party-1 supporters : 240 (48.00%)
9     Party-2 supporters : 160 (32.00%)
10    Other voters       : 100 (20.00%)
11    *----- random polls -----
12    Party_1(48.0%) | Party_2(32.0%) | expected
13    -----
14    a2 : 30.84%   | a1 : 30.84%   | a2 : 15.56%
15    a3 : 23.67%   | a4 : 23.67%   | a3 : 12.91%
16    a7 : 17.29%   | a6 : 17.29%   | a7 : 11.43%
17    a5 : 11.22%   | a5 : 11.22%   | a1 : 11.00%
18    a6 : 09.79%   | a7 : 09.79%   | a6 : 10.23%
19    a4 : 04.83%   | a3 : 04.83%   | a4 : 09.89%
20    a1 : 02.37%   | a2 : 02.37%   | a5 : 08.98%
21 >>> lvp.save2PerfTab('divisiveExample')
22 >>> dvp = PerformanceTableau('divisiveExample')
23 >>> from outrankingDigraphs import \
24 ...     UnOpposedBipolarOutrankingDigraph
25 >>> uodg = UnOpposedBipolarOutrankingDigraph(dvp)
26 >>> uodg
27    *----- Object instance description -----*
28    Instance class : UnOpposedBipolarOutrankingDigraph
29    Instance name  : unopposed_outrankings
30    Actions        : 7
31    Criteria       : 500
32    Size           : 0
33    Oppositeness (%) : 100.00
34    Determinateness (%) : 50.00
35    Valuation domain : [-1.00;1.00]
```

With an oppositeness degree of 100.0% (see Listing 20.6 Lines 31-32), 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)
2    * ---- Relation Table -----
3    r   | 'a1'  'a2'  'a3'  'a4'  'a5'  'a6'  'a7'
```

```

4   -----| -----
5   'a1' | - +0.00 +0.00 +0.00 +0.00 +0.00 +0.00
6   'a2' | +0.00 - +0.00 +0.00 +0.00 +0.00 +0.00
7   'a3' | +0.00 +0.00 - +0.00 +0.00 +0.00 +0.00
8   'a4' | +0.00 +0.00 +0.00 - +0.00 +0.00 +0.00
9   'a5' | +0.00 +0.00 +0.00 +0.00 - +0.00 +0.00
10  'a6' | +0.00 +0.00 +0.00 +0.00 +0.00 - +0.00
11  'a7' | +0.00 +0.00 +0.00 +0.00 +0.00 +0.00 -
12 Valuation domain: [-1.0; 1.0]

```

As a consequence, a multipartisan primary selection, computed with a `showBestChoiceRecommendation()` method, will keep the complete initial set of eligible candidates and, hence, becomes *ineffective* (see Listing 20.7 Line 6).

Listing 20.7 Example of ineffective primary multipartisan selection

```

1 >>> uodg.showBestChoiceRecommendation()
2 Rubis best choice recommendation(s) (BCR)
3 (in decreasing order of determinateness)
4 Credibility domain: [-1.00,1.00]
5 === >> ambiguous choice(s)
6 choice : ['a1','a2','a3','a4','a5','a6','a7']
7 independence : 0.00
8 dominance : 1.00
9 absorbency : 1.00
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 below, we see, for instance, that the simple majority winner is 'a2' (Line 2), whereas the instant-run-off winner is 'a6' (Line 4).

Listing 20.8 Example of non obvious secondary selection

```

1 >>> lvp.computeSimpleMajorityWinner()
2 ['a2']
3 >>> lvp.computeInstantRunoffWinner()
4 ['a6']
5 >>> from votingProfiles import MajorityMarginsDigraph
6 >>> cg = MajorityMarginsDigraph(lvp)
7 >>> cg.showRelationTable(ReflexiveTerms=False)
8 * ----- Relation Table -----
9 r() | 'a1' 'a2' 'a3' 'a4' 'a5' 'a6' 'a7'
10 -----|-----
11 'a1' | - -68 -90 -46 -68 -88 -84
12 'a2' | +68 - -32 +80 +46 -6 -24
13 'a3' | +90 +32 - +58 +46 +4 +8
14 'a4' | +4 -80 -58 - -16 -68 -72
15 'a5' | +68 -46 -46 +16 - -26 -64
16 'a6' | +88 +6 -4 +68 +26 - -2
17 'a7' | +84 +24 -8 +72 +64 +2 -
18 Valuation domain: [-500;+500]
19 >>> cg.computeCondorcetWinners()
20 ['a3']
21 >>> lvp.computeBordaWinners()

```

```

22      ['a3','a7']
23 >>> cg.computeCopelandRanking()
24      ['a3', 'a7', 'a6', 'a2', 'a5', 'a4', 'a1']

```

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 13,20), 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).

We may eventually visualize in Fig. 20.2 this linear ranking with a graphviz drawing where we drop all transitive arcs (Line 1) and orient the drawing with CONDORCET winner 'a3' and loser 'a1' (Line 2 below).

```

1 >>> cg.closeTransitive(Reverse=True)
2 >>> cg.exportGraphViz('divGraph', \
3 ...           bestChoice=['a3'],worstChoice=['a1'])
4     *----- exporting a dot file for GraphViz tools -----*
5     Exporting to divGraph.dot
6     dot -Grankdir=BT -Tpng divGraph.dot -o divGraph.png

```

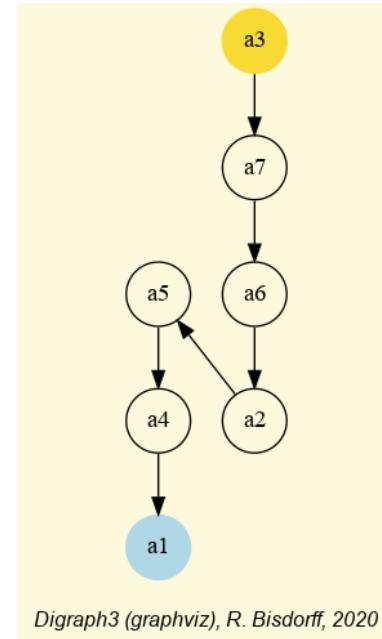


Fig. 20.2 The linear ranking modelled by the majority margins digraph.

Digraph3 (graphviz), R. Bisendorff, 2020

The last Chapter 21 propose eventually a bipolar approval voting system which could help healing a society who is facing important social choices in an excessive divisive political system.

Chapter 21

Tempering plurality tyranny effects with bipolar approval voting

Abstract

The choice of a voting procedure shapes the democracy in which we live.

– Baujard A., Gavrel F., Iggersheim H., Laslier J.-F. and Lebon I. [BAU-2013p].

21.1 Bipolar approval voting systems

In the `votingProfiles` module we provide a `BipolarApprovalVotingProfile` class for handling voting results where, for each eligible candidate c , the voters are invited to *approve* (+1), *disapprove* (-1), or *ignore* (0) the statement that candidate c should win the election.

File `bpApVotingProfile.py` Footnote[x] contains such a bipolar approval voting profile concerning 100 voters and 15 eligible candidates. We may inspect its content as follows.

```
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']
```

Beside the `candidates` and `voters` attributes, we discover in Line 11 above the `approvalBallot` attribute which gathers bipolar approval votes. Its content is the following.

Listing 21.1 Inspecting a bipolar approval ballot

```
1 >>> bavp.approvalBallot
```

```

2     {'v001':
3         {'a01': Decimal('0'),
4          ...
5          'a04': Decimal('1'),
6          ...
7          'a15': Decimal('0')
8      },
9     'v002':
10    {'a01': Decimal('-1'),
11     'a02': Decimal('0'),
12     ...
13     'a15': Decimal('1')
14 },
15 ...
16     'v100':
17     {'a01': Decimal('0'),
18     'a02': Decimal('1'),
19     ...
20     'a15': Decimal('1')
21   }
22 }
```

Let us denote A_v the set of candidates approved by voter v . In the `approvalBallot` attribute we hence record in fact the bipolar-valued truth characteristic values $r(c \in A_v)$ of the statements that candidate c is *approved* by voter v . In Listing 21.1 Line 5, we observe for instance that voter 'v001' positively approves candidate 'a04'. And, in Line 10, we see that voter 'v002' negatively approves, i.e. positively disapproves candidate 'a01'.

We may now consult how many approvals or disapprovals each candidate receives.

```

1 >>> bavp.showApprovalResults()
2     Approval results
3     Candidate: 'a12' obtains 34 votes
4     Candidate: 'a05' obtains 30 votes
5     Candidate: 'a03' obtains 28 votes
6     Candidate: 'a14' obtains 27 votes
7     Candidate: 'a11' obtains 27 votes
8     Candidate: 'a04' obtains 27 votes
9     Candidate: 'a01' obtains 27 votes
10    Candidate: 'a13' obtains 24 votes
11    Candidate: 'a07' obtains 24 votes
12    Candidate: 'a15' obtains 23 votes
13    Candidate: 'a02' obtains 23 votes
14    Candidate: 'a09' obtains 22 votes
15    Candidate: 'a08' obtains 22 votes
16    Candidate: 'a10' obtains 21 votes
17    Candidate: 'a06' obtains 21 votes
18    Total approval votes: 380
19    Approval proportion: 380/1500 = 0.25
20 >>> bavp.showDisapprovalResults()
21     Disapproval results
22     Candidate: 'a12' obtains 16 votes
```

```

23 Candidate: 'a03' obtains 22 votes
24 Candidate: 'a09' obtains 23 votes
25 Candidate: 'a04' obtains 24 votes
26 Candidate: 'a06' obtains 24 votes
27 Candidate: 'a13' obtains 24 votes
28 Candidate: 'a11' obtains 25 votes
29 Candidate: 'a02' obtains 26 votes
30 Candidate: 'a07' obtains 26 votes
31 Candidate: 'a08' obtains 26 votes
32 Candidate: 'a05' obtains 27 votes
33 Candidate: 'a10' obtains 27 votes
34 Candidate: 'a14' obtains 27 votes
35 Candidate: 'a15' obtains 27 votes
36 Candidate: 'a01' obtains 32 votes
37 Total disapproval votes: 376
38 Disapproval proportion: 376/1500 = 0.25

```

In Lines 3 and 22 above, we may see that, of all potential candidates, it is Candidate 'a12' who receives the highest number of approval votes (34) and the lowest number of disapproval votes (16). Total number of approval, respectively disapproval, votes approaches more or less a proportion of 25% of the $100 \times 15 = 1500$ potential approval votes. About 50% of the latter remain hence ignored.

When operating now, for each candidate c , the difference between the number of approval and the number of disapproval votes he receives, we obtain per candidate a corresponding *net approval* score; in fact, the bipolar truth characteristic value of the statement "*candidate c should win the election*".

$$r(\text{Candidate } c \text{ should win the election}) = \sum_v (r(c \in A_v)). \quad (21.1)$$

These bipolar characteristic values are stored in the `netApprovalScores` attribute and may be printed out as follows:

```

1 >>> bavp.showNetApprovalScores()
2 Net Approval Scores
3 Candidate: 'a12' obtains 18 net approvals
4 Candidate: 'a03' obtains 6 net approvals
5 Candidate: 'a05' obtains 3 net approvals
6 Candidate: 'a04' obtains 3 net approvals
7 Candidate: 'a11' obtains 2 net approvals
8 Candidate: 'a14' obtains 0 net approvals
9 Candidate: 'a13' obtains 0 net approvals
10 Candidate: 'a09' obtains -1 net approvals
11 Candidate: 'a07' obtains -2 net approvals
12 Candidate: 'a06' obtains -3 net approvals
13 Candidate: 'a02' obtains -3 net approvals
14 Candidate: 'a15' obtains -4 net approvals
15 Candidate: 'a08' obtains -4 net approvals
16 Candidate: 'a01' obtains -5 net approvals
17 Candidate: 'a10' obtains -6 net approvals

```

We observe in Line 3 above that Candidate 'a12', with a net approval score of $34 - 16 = 18$, represents indeed the *best approved* candidate for winning the elec-

tion. With a net approval score of $28 - 22 = 6$, Candidate 'a03' appears 2nd-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).

21.2 Pairwise comparison of bipolar approval votes

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.

```

1 >>> bavp.ballot['v001']['a12']
2   {'a02': Decimal('1'), 'a11': Decimal('1'),
3    'a14': Decimal('1'), 'a04': Decimal('0'),
4    'a06': Decimal('1'), 'a05': Decimal('1'),
5    'a12': Decimal('0'), 'a13': Decimal('0'),
6    'a15': Decimal('1'), 'a01': Decimal('1'),
7    'a08': Decimal('1'), 'a07': Decimal('1'),
8    'a09': Decimal('0'), 'a03': Decimal('1'),
9    'a10': Decimal('0')}

```

For voter 'v001, for instance, the best approved candidate 'a12' is strictly preferred to candidates: 'a01', 'a02', 'a03', 'a05', 'a06', 'a07', 'a08', 'a11', 'a14' and '15'. No candidate is preferred to 'a12' and the comparison with 'a04', 'a09', 'a10' and 'a13' is not communicated, hence indeterminate. Mind by the way that the reflexive comparison of 'a12' with itself is, as usual, is ignored, i.e. indeterminate. Each voter v defines thus a partially determined transitive strict preference relation denoted \succ_v on the eligible candidates.

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 $r(x \succ y)$:

$$r(x \succ y) = \sum_v (r(x \succ_v y)) . \quad (21.2)$$

We say that candidate x is *better approved than* Candidate y when $r(x \succ y) > 0$, 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 $r(x \succ y) < 0$, 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
2 >>> m = MajorityMarginsDigraph(bavp)
3 >>> m
4   *----- Digraph instance description -----*

```

```

5   Instance class      : MajorityMarginsDigraph
6   Instance name       : rel_bpApVotingProfile
7   Digraph Order       : 15
8   Digraph Size        : 97
9   Valuation domain    : [-100.00;100.00]
10  Determinateness (%) : 52.55
11  Attributes          : ['name', 'actions',
12    'criteria', 'ballot',
13    'valuationdomain', 'relation',
14    'order', 'gamma', 'notGamma']

```

The resulting digraph m contains 97 positively validated relations (see Line 8 above) and (see Line 9) for all pairs (x,y) of eligible candidates, $r(x \succ y)$ takes value in an valuation range from -100.00 (all voters opposed) to $+100.00$ (unanimously supported).

We may inspect these pairwise $r(x \succ y)$ values in a browser view:

```
>>> m.showHTMLRelationTable(relationName='r(x > y)')
```

$r(x \geq y)$	a01	a02	a03	a04	a05	a06	a07	a08	a09	a10	a11	a12	a13	a14	a15
a01	-	1	-10	-10	-8	-2	-2	-3	-3	2	-5	-19	-4	-2	2
a02	-1	-	-5	-4	-9	-2	5	1	-7	3	-2	-20	-1	-4	-2
a03	10	5	-	1	4	6	5	10	7	11	0	-5	2	6	9
a04	10	4	-1	-	2	5	7	8	-1	4	0	-7	8	2	8
a05	8	9	-4	-2	-	4	6	7	4	2	1	-17	6	0	3
a06	2	2	-6	-5	-4	-	-1	2	-3	5	-4	-13	-1	-1	2
a07	2	-5	-5	-7	-6	1	-	7	-4	2	-5	-17	-2	-2	4
a08	3	-1	-10	-8	-7	-2	-7	-	-2	2	-2	-16	0	0	-1
a09	3	7	-7	1	-4	3	4	2	-	3	0	-18	-4	0	2
a10	-2	-3	-11	-4	-2	-5	-2	-2	-3	-	-6	-15	-4	-4	2
a11	5	2	0	0	-1	4	5	2	0	6	-	-15	4	0	5
a12	19	20	5	7	17	13	17	16	18	15	15	-	12	13	18
a13	4	1	-2	-8	-6	1	2	0	4	4	-4	-12	-	1	4
a14	2	4	-6	-2	0	1	2	0	0	4	0	-13	-1	-	-1
a15	-2	2	-9	-8	-3	-2	-4	1	-2	-2	-5	-18	-4	1	-

Valuation domain: [-100; +100]

Fig. 21.1 The bipolar-valued pairwise majority margins

It gets easily apparent that candidate 'a12' 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, a linear ranking of the 15 eligible candidates and compare the result with the net approval scores ranking.

Listing 21.2 Comparing the net approval and the NETFLOWS rankings

```

1 >>> from linearOrders import NetFlowsOrder
2 >>> nf = NetFlowsOrder(m,Comments=True)
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 %+.1f' %
7 ...           (item[1], item[0],\
8 ...            bavp.netApprovalScores[item[1]]))
9
10 NetFlows versus Net Approval Ranking
11 Candidate    NetFlows score   Net Approval score
12   'a12'        +410.000      +18.0
13   'a03'        +142.000      +6.0
14   'a04'        +98.000       +3.0
15   'a05'        +54.000       +3.0
16   'a11'        +34.000       +2.0
17   'a09'        -16.000       -1.0
18   'a14'        -20.000       +0.0
19   'a13'        -22.000       +0.0
20   'a06'        -50.000       -3.0
21   'a07'        -74.000       -2.0
22   'a02'        -96.000       -3.0
23   'a08'        -102.000      -4.0
24   'a15'        -110.000      -4.0
25   'a10'        -122.000      -6.0
      'a01'        -126.000      -5.0

```

In Listing 21.2 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 'a09' advances before candidates 'a13' and 'a14' (see Line 16), and 'a6' and 'a07' swap their positions 9 and 10 (see Line 19). And, the two last-ranked candidates 'a10' and 'a01' also swap their positions.

This confirms again the pertinence of the net approval scoring approach for finding the winner in a bipolar approving voting system. Yet, voting by approving (+1), disapproving (-1) or ignoring (0) eligible candidates, may also be seen as a performance evaluation of the eligible candidates on a $\{-1, 0, 1\}$ -graded ordinal scale.

21.3 Three-valued evaluative voting system

Following such an epistemic perspective, we may effectively convert the given `BipolarApprovalVotingProfile` instance into a `PerformanceTableau` instance, so as to get access to a corresponding outranking decision aiding approach.

Mind that, contrary to the majority margins of the “*better approved than*” relation, all voters consider now the approved candidates to be all equivalent (+1). Same is true for the disapproved (-1), respectively the ignored (0) candidates. The voter’s

marginal preferences model this time a complete preorder with three equivalence classes.

From the saved file `AVPerfTab.py` (see Line 1 below), we may construct an outranking relation on the eligible candidates with our standard `BipolarOutrankingDigraph` constructor. The semantics of this outranking relation are the following:

- We say that Candidate x *outranks* Candidate y when there is a majority of voters who consider x *at least as well evaluated as* y (see Line 3 below);
- We say that Candidate x is *outranked by* Candidate y when there is a majority of voters who consider x *not at least as well evaluated as* y .

Listing 21.3 Computing the outranking digraph

```

1 >>> bavp.save2PerfTab(fileName='AVPerfTab', valueDigits=0)
2     *--- Saving as performance tableau in AVPerfTab.py ---*
3 >>> from outrankingDigraphs import \
4             BipolarOutrankingDigraph
5 >>> odg = BipolarOutrankingDigraph('AVPerfTab')
6 >>> odg
7     *----- Object instance description -----*
8     Instance class      : BipolarOutrankingDigraph
9     Instance name       : rel_AVPerfTab
10    Actions            : 15
11    Criteria           : 100
12    Size               : 210
13    Determinateness (%) : 69.29
14    Valuation domain   : [-1.00;1.00]
15    Attributes         : ['name', 'actions', 'order',
16                           'criteria', 'evaluation', 'NA',
17                           'valuationsdomain', 'relation',
18                           'gamma', 'notGamma', ...]
```

The size ($210 = 15 \times 14$) of the resulting outranking digraph odg , shown in Listing 21.3 Line 11 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 21.4 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   'a12        +4.100      +18.0
12   'a03        +1.420      +6.0
13   'a04        +0.980      +3.0
14   'a05        +0.540      +3.0
15   'a11        +0.340      +2.0
16   'a09        -0.160      -1.0
17   'a14        -0.200      +0.0
18   'a13        -0.220      +0.0
19   'a06        -0.500      -3.0
20   'a07        -0.740      -2.0
21   'a02        -0.960      -3.0
22   'a08        -1.020      -4.0
23   'a15        -1.100      -4.0
24   'a10        -1.220      -6.0
25   'a01        -1.260      -5.0

```

Despite its apparent poor strict preference discriminating power, we obtain in Listing 21.4 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 21.2).

Encouraged by this positive result, we may furthermore try to compute as well a best choice recommendation.

Listing 21.5 Computing a best social choice recommendation

```

1 >>> odg.showBestChoiceRecommendation(CoDual=True)
2 Rubis best choice recommendation(s) (BCR)
3 (in decreasing order of determinateness)
4 Credibility domain: [-1.00,1.00]
5 === >> ambiguous best choice(s)
6   * choice      : ['a01','a02','a03','a04','a05',
7     'a06','a07','a08','a09','a10',
8     'a11','a12','a13','a14','a15']
9   independence  : 0.06
10  dominance    : 1.00
11  absorbency   : 1.00
12  covering (%) : 100.00
13  determinateness (%) : 61.13
14  - most credible action(s) = {
15    'a12': 0.44, 'a03': 0.34, 'a04': 0.30,
16    'a14': 0.28, 'a13': 0.24, 'a06': 0.24,
17    'a11': 0.20, 'a10': 0.20, 'a07': 0.20,
18    'a01': 0.20, 'a08': 0.18, 'a05': 0.18,
19    'a15': 0.14, 'a09': 0.14, 'a02': 0.06, }
20 === >> ambiguous worst choice(s)
21   * choice      : ['a01','a02','a03','a04','a05',
22     'a06','a07','a08','a09','a10',
23     'a11','a12','a13','a14','a15']
24   independence  : 0.06
25   dominance    : 1.00

```

```

26     absorbency      : 1.00
27     covered (%)    : 100.00
28     determinateness (%) : 63.73
29     - most credible action(s) = {
30         'a13': 0.36, 'a06': 0.36, 'a15': 0.34,
31         'a01': 0.34, 'a08': 0.32, 'a07': 0.30,
32         'a02': 0.30, 'a14': 0.28, 'a11': 0.28,
33         'a09': 0.28, 'a04': 0.26, 'a10': 0.24,
34         'a05': 0.20, 'a03': 0.20, 'a12': 0.06, }
```

The strict outranking digraph ($\sim (-odg)$) being actually *empty*, we obtain a unique *ambiguous* –best as well as last– choice recommendation which trivially retains all fifteen candidates (see Listing 21.5 Lines 6-8 above). Yet, the bipolar-valued best choice membership characteristic vector reveals that, among all the fifteen potential winners, it is indeed Candidate 'a12' the most credible one with a 72% majority of voters' support (see Line 15, $(0.44 + 1.0)/2 = 0.72$); followed by Candidate 'a03' (67%) and Candidate 'a04' (65%). Similarly, Candidates 'a13' and 'a06' represent the most credible losers with a 68% majority voters' support (Line 30).

We observe here empirically that *evaluative* voting systems, using three-valued ordinal performance scales, match closely a corresponding bipolar approval voting systems. The latter voting system models, however, more faithfully the very preferential information that is expressed with *approved*, *disapproved* or *ignored* statements. The corresponding evaluation on a three-graded scale, being value (numbers) based, cannot express the fact that in bipolar approval voting systems there is no preferential information given concerning the pairwise comparison of all approved, respectively disapproved or ignored candidates.

Let us finally illustrate how bipolar approval voting systems may favour multipartisan supported candidates. We shall therefore compare *bipolar approval* versus *uninominal plurality* election results when considering a highly divisive and partisan political context.

21.4 Favouring multipartisan candidates

In modern democracy, politics are largely structured by political parties and activists movements. Let us so consider a bipolar approval voting profile *dvp* where the random voter behaviour is simulated from two pre-electoral polls concerning a political scene with essentially two major competing parties, like the one existing in the US.

Listing 21.6 A random bipolar approval voting profile in a divisive political context

```

1 >>> dvp = RandomBipolarApprovalVotingProfile(\n2 ...                 numberOfCandidates=15,\n3 ...                 numberOfVoters=100,\n4 ...                 approvalProbability=0.25,\n5 ...                 disapprovalProbability=0.25,\n6 ...                 WithPolls=True,\n7 ...                 partyRepartition=0.5,\n8 ...                 other=0.05,
```

```

9 ...             DivisivePolitics=True,
10 ...             seed=200)
11 >>> dvp.showRandomPolls()
12 Random repartition of voters
13 Party-1 supporters : 45 (45.00%)
14 Party-2 supporters : 49 (49.00%)
15 Other voters       : 6 (06.00%)
16 ----- random polls -----
17 Party-1(45.0%) | Party-2(49.0%) | expected
18 -----
19 'a05' : 24.10% | 'a07' : 24.10% | 'a07' : 11.87%
20 'a14' : 23.48% | 'a10' : 23.48% | 'a10' : 11.60%
21 'a03' : 15.13% | 'a01' : 15.13% | 'a05' : 10.91%
22 'a12' : 07.55% | 'a04' : 07.55% | 'a14' : 10.67%
23 'a08' : 07.11% | 'a09' : 07.11% | 'a01' : 07.67%
24 'a15' : 04.37% | 'a13' : 04.37% | 'a03' : 07.09%
25 'a11' : 03.99% | 'a02' : 03.99% | 'a04' : 04.55%
26 'a06' : 03.80% | 'a06' : 03.80% | 'a09' : 04.49%
27 'a02' : 02.79% | 'a11' : 02.79% | 'a12' : 04.32%
28 'a13' : 02.63% | 'a15' : 02.63% | 'a08' : 04.30%
29 'a09' : 02.24% | 'a08' : 02.24% | 'a06' : 03.57%
30 'a04' : 01.89% | 'a12' : 01.89% | 'a13' : 03.32%
31 'a01' : 00.57% | 'a03' : 00.57% | 'a15' : 03.25%
32 'a10' : 00.20% | 'a14' : 00.20% | 'a02' : 03.21%
33 'a07' : 00.14% | 'a05' : 00.14% | 'a11' : 03.16%

```

In Listing 21.6, the divisive political situation is reflected by the fact that Party-1 and Party-2 supporters show strict reversed preferences. The leading candidates of Party-1 ('a05' and 'a14') are last choices for Party-2 supporters and, Candidates 'a07' and 'a10', leading candidates for Party-2 supporters, are similarly the least choices for Party-1 supporters.

No clear winner may be guessed from these pre-election polls. As Party-2 shows however slightly more supporters than Party-1, the expected winner in an uninominal plurality or instant-runoff voting system will be Candidate 'a07', i.e, the leading candidate of majority Party-2 (see below).

```

1 >>> dvp.computeSimpleMajorityWinner()
2 ['a07']
3 >>> dvp.computeInstantRunoffWinner()
4 ['a07']

```

Now, in a corresponding bipolar approval voting system, Party-1 supporters will usually approve their leading candidates and disapprove the leading candidates of Party-2. Vice versa, Party-2 supporters will usually approve their leading candidates and disapprove the leading candidates of Party-1. Let us consult the resulting approval votes per candidate.

```

1 >>> dvp.showApprovalResults()
2     Candidate: 'a07' obtains 30 votes
3     Candidate: 'a10' obtains 28 votes
4     Candidate: 'a05' obtains 28 votes
5     Candidate: 'a01' obtains 28 votes
6     Candidate: 'a03' obtains 26 votes

```

```

7 Candidate: 'a02' obtains 26 votes
8 Candidate: 'a12' obtains 25 votes
9 Candidate: 'a14' obtains 24 votes
10 Candidate: 'a13' obtains 24 votes
11 Candidate: 'a09' obtains 21 votes
12 Candidate: 'a04' obtains 21 votes
13 Candidate: 'a08' obtains 19 votes
14 Candidate: 'a06' obtains 17 votes
15 Candidate: 'a15' obtains 15 votes
16 Candidate: 'a11' obtains 12 votes
17 Total approval votes: 344
18 Approval proportion: 344/1500 = 0.23

```

When considering only the approval votes, we find confirmed above that the leading candidate of Party-2 obtains in this simulation a plurality of approval votes. In uninominal plurality or instant-runoff voting systems, this candidate wins hence the election, quite to the despair of Party-1 supporters. As a foreseeable consequence, this election result will be more or less aggressively contested which leads to a loss of popular trust in democratic elections and institutions.

If we look however on the corresponding disapprovals, we discover that, not surprisingly, the leading candidates of both parties collect by far the highest number of disapproval votes.

```

1 >>> dvp.showDisapprovalResults()
2     Candidate: 'a02' obtains 14 votes
3     Candidate: 'a04' obtains 14 votes
4     Candidate: 'a13' obtains 14 votes
5     Candidate: 'a06' obtains 15 votes
6     Candidate: 'a09' obtains 15 votes
7     Candidate: 'a08' obtains 16 votes
8     Candidate: 'a11' obtains 16 votes
9     Candidate: 'a15' obtains 18 votes
10    Candidate: 'a12' obtains 20 votes
11    Candidate: 'a01' obtains 29 votes
12    Candidate: 'a03' obtains 30 votes
13    Candidate: 'a10' obtains 37 votes
14    Candidate: 'a07' obtains 44 votes
15    Candidate: 'a14' obtains 45 votes
16    Candidate: 'a05' obtains 49 votes
17    Total disapproval votes: 376
18    Disapproval proportion: 376/1500 = 0.25

```

Balancing now approval against disapproval votes will favour the moderate, bipartisan supported, candidates.

```

1 >>> dvp.showNetApprovalScores()
2     Net Approval Scores
3     Candidate: 'a02' obtains 12 net approvals
4     Candidate: 'a13' obtains 10 net approvals
5     Candidate: 'a04' obtains 7 net approvals
6     Candidate: 'a09' obtains 6 net approvals
7     Candidate: 'a12' obtains 5 net approvals
8     Candidate: 'a08' obtains 3 net approvals
9     Candidate: 'a06' obtains 2 net approvals

```

```

10 Candidate: 'a01' obtains -1 net approvals
11 Candidate: 'a15' obtains -3 net approvals
12 Candidate: 'a11' obtains -4 net approvals
13 Candidate: 'a03' obtains -4 net approvals
14 Candidate: 'a10' obtains -9 net approvals
15 Candidate: 'a07' obtains -14 net approvals
16 Candidate: 'a14' obtains -21 net approvals
17 Candidate: 'a05' obtains -21 net approvals

```

Candidate 'a02', 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 21.6), shows indeed the highest net approval score. Second highest net approval score obtains Candidate 'a13', in position 6 for Party-2 and in position 10 for Party-1 supporters.

Fig. 21.2, showing the NETFLOWS ranked relation table of the “*better approved than*” majority margins digraph, confirms below this net approval scoring result.

```

1 >>> m = MajorityMarginsDigraph(dvp)
2 >>> m.showHTMLRelationTable(\n3 ...     actionsList=m.computeNetFlowsRanking(),\n4 ...     relationName='r(x > y)')

```

r(x > y)	a02	a13	a04	a09	a12	a08	a06	a01	a11	a15	a03	a10	a07	a14	a05
a02	-	6	5	6	9	5	10	12	12	14	11	15	22	22	23
a13	-6	-	2	5	2	5	8	10	14	10	9	13	18	23	20
a04	-5	-2	-	0	2	2	3	7	7	8	11	13	18	21	18
a09	-6	-5	0	-	0	2	5	5	11	9	5	13	16	21	16
a12	-9	-2	-2	0	-	4	6	2	9	6	10	5	10	23	25
a08	-5	-5	-2	-2	-4	-	4	0	8	9	7	5	11	21	20
a06	-10	-8	-3	-5	-6	-4	-	-2	5	5	5	6	13	17	18
a01	-12	-10	-7	-5	-2	0	2	-	1	-1	3	8	11	9	13
a11	-12	-14	-7	-11	-9	-8	-5	-1	-	1	-2	7	14	13	16
a15	-14	-10	-8	-9	-6	-9	-5	1	-1	-	0	3	10	14	14
a03	-11	-9	-11	-5	-10	-7	-5	-3	2	0	-	-3	7	16	16
a10	-15	-13	-13	-13	-5	-5	-6	-8	-7	-3	3	-	3	6	7
a07	-22	-18	-18	-16	-10	-11	-13	-11	-14	-10	-7	-3	-	3	5
a14	-22	-23	-21	-21	-23	-21	-17	-9	-13	-14	-16	-6	-3	-	0
a05	-23	-20	-18	-16	-25	-20	-18	-13	-16	-14	-16	-7	-5	0	-

Valuation domain: [-100; +100]

Fig. 21.2 The pairwise *better approved than* majority margins

Candidate 'a02' appears indeed *better approved than* any other candidate (CONDORCET winner); and, the leading candidates of Party-1, 'a05' and 'a14', are *less approved than* any other candidates (weak CONDORCET losers).

```

1 >>> m.computeCondorcetWinners()

```

```

2      ['a02']
3 >>> m.computeWeakCondorcetLosers()
4      ['a05','a14']

```

We see this result furthermore confirmed when computing the corresponding best, respectively worst choice recommendation.

```

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 best choice(s)
6   * choice          : ['a02']
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) = { 'a02': 5.00, }
13 === >> potential worst choice(s)
14   * choice          : ['a05', 'a14']
15     independence    : 0.00
16     dominance       : -23.00
17     absorbency      : 5.00
18     covered (%)     : 100.00
19     determinateness (%) : 50.00
20     - most credible action(s) = { }

```

Candidate 'a02', being actually a CONDORCET winner, gives an initial prekernel of digraph m , whereas Party-1 leading Candidates 'a05' and 'a14', both being weak CONDORCET losers, give together a terminal prekernel. They hence represent our *best choice*, respectively, *worst choice* recommendations for winning this simulated election.

Let us conclude by predicting that, for leading political candidates in an aggressively divisive political context, the perspective to easily fail election with bipolar approval voting systems, 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's do it !*.

It is worthwhile noticing again the essential structural and computational role, the ignored value is again playing in bipolar approval 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.

Part V
Working with simple graphs

To be written ...

Chapter 22

Working with undirected graphs

Abstract

22.1 Implementing undirected graphs

In the DIGRAPH3graphs module, the root Graph class provides a generic simple graph model, without loops and multiple links. A given object of this class contains at least the following attributes in:

1. vertices: a dictionary of vertices with name and shortName attributes,
2. edges : a dictionary with *frozensets* Footnote[x] of pairs of vertices as entries carrying a characteristic value in the range of the previous valuation domain,
3. valuationDomain: a dictionary with three entries: the minimum (-1, means certainly no link), the median (0, means missing information) and the maximum characteristic value (+1, means certainly a link),
4. gamma: a dictionary containing the direct neighbors of each vertex, automatically added by the object constructor.

Example Python3 session:

```
1 >>> from graphs import Graph
2 >>> g = Graph(numberOfVertices=7, edgeProbability=0.5)
3 >>> g.save(fileName='tutorialGraph')
```

The saved Graph instance named tutorialGraph.py is encoded as follows.

```
1 # Graph instance saved in Python format
2 vertices = {
3     'v1': {'shortName': 'v1', 'name': 'random vertex'},
4     'v2': {'shortName': 'v2', 'name': 'random vertex'},
5     'v3': {'shortName': 'v3', 'name': 'random vertex'},
6     'v4': {'shortName': 'v4', 'name': 'random vertex'},
7     'v5': {'shortName': 'v5', 'name': 'random vertex'},
```

```

8   'v6': {'shortName': 'v6', 'name': 'random vertex'},
9   'v7': {'shortName': 'v7', 'name': 'random vertex'},
10  }
11 valuationDomain = {'min':-1,'med':0,'max':1}
12 edges = {
13   frozenset(['v1','v2']) : -1,
14   frozenset(['v1','v3']) : -1,
15   frozenset(['v1','v4']) : -1,
16   frozenset(['v1','v5']) : 1,
17   frozenset(['v1','v6']) : -1,
18   frozenset(['v1','v7']) : -1,
19   frozenset(['v2','v3']) : 1,
20   frozenset(['v2','v4']) : 1,
21   frozenset(['v2','v5']) : -1,
22   frozenset(['v2','v6']) : 1,
23   frozenset(['v2','v7']) : -1,
24   frozenset(['v3','v4']) : -1,
25   frozenset(['v3','v5']) : -1,
26   frozenset(['v3','v6']) : -1,
27   frozenset(['v3','v7']) : -1,
28   frozenset(['v4','v5']) : 1,
29   frozenset(['v4','v6']) : -1,
30   frozenset(['v4','v7']) : 1,
31   frozenset(['v5','v6']) : 1,
32   frozenset(['v5','v7']) : -1,
33   frozenset(['v6','v7']) : -1,
34 }
```

The stored graph can be reloaded and plotted with the generic `exportGraphViz()` Footnote[1] method as follows:

```

1 >>> g = Graph('tutorialGraph')
2 >>> g.exportGraphViz()
3 *----- exporting a dot file for GraphViz tools -----
4 Exporting to tutorialGraph.dot
5 fdp -Tpng tutorialGraph.dot -o tutorialGraph.png
```

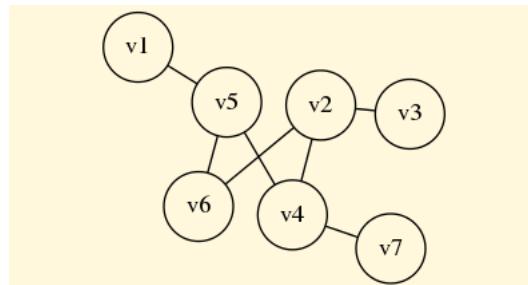


Fig. 22.1 Example simple graph instance

Graphs Python module (graphviz), R. Bisдорff, 2011

Properties, like the gamma function and vertex degrees and neighbourhood depths may be shown with a `showShort()` method.

```

1 >>> g.showShort()
2     *---- short description of the graph ----*
3     Name : 'tutorialGraph'
4     Vertices : ['v1','v2','v3','v4','v5','v6','v7']
5     Valuation domain : {'min': -1, 'med': 0, 'max': 1}
6     Gamma function :
7         v1 -> ['v5']
8         v2 -> ['v6', 'v4', 'v3']
9         v3 -> ['v2']
10        v4 -> ['v5', 'v2', 'v7']
11        v5 -> ['v1', 'v6', 'v4']
12        v6 -> ['v2', 'v5']
13        v7 -> ['v4']
14        degrees : [0, 1, 2, 3, 4, 5, 6]
15        distribution : [0, 3, 1, 3, 0, 0, 0]
16        nbh depths : [0, 1, 2, 3, 4, 5, 6, 'inf.']
17        distribution : [0, 0, 1, 4, 2, 0, 0, 0]
```

A `Graph` instance corresponds bijectively to a symmetric `Digraph` instance and we may easily convert from one to the other with the `graph2Digraph()`, and vice versa with the `digraph2Graph()` method. Thus, all computing resources of the `Digraph` class, suitable for symmetric digraphs, become readily available, and vice versa.

```

1 >>> dg = g.graph2Digraph()
2 >>> dg.showRelationTable(ndigits=0,ReflexiveTerms=False)
3     * ---- Relation Table ----
4     S | 'v1'  'v2'  'v3'  'v4'  'v5'  'v6'  'v7'
5     -----|-----
6     'v1' |   -    -1    -1    -1    1    -1    -1
7     'v2' |   -1    -    1    1    -1    1    -1
8     'v3' |   -1    1    -    -1    -1    -1    -1
9     'v4' |   -1    1    -1    -    1    -1    1
10    'v5' |   1    -1    -1    1    -    1    -1
11    'v6' |   -1    1    -1    -1    1    -    -1
12    'v7' |   -1    -1    -1    1    -1    -1    -
13 >>> g1 = dg.digraph2Graph()
14 >>> g1.showShort()
15     *---- short description of the graph ----*
16     Name      : 'tutorialGraph'
17     Vertices  : ['v1','v2','v3','v4','v5','v6','v7']
18     Valuation domain : {'med': 0, 'min': -1, 'max': 1}
19     Gamma function :
20         v1 -> ['v5']
21         v2 -> ['v3', 'v6', 'v4']
22         v3 -> ['v2']
23         v4 -> ['v5', 'v7', 'v2']
24         v5 -> ['v6', 'v1', 'v4']
25         v6 -> ['v5', 'v2']
26         v7 -> ['v4']
27        degrees : [0, 1, 2, 3, 4, 5, 6]
```

```

28     distribution : [0, 3, 1, 3, 0, 0, 0]
29     nbh depths  : [0, 1, 2, 3, 4, 5, 6, 'inf.']
30     distribution : [0, 0, 1, 4, 2, 0, 0]

```

22.2 q-coloring of a graph

A 3-coloring of the tutorial graph g may for instance be computed and plotted with the `Q_Coloring` class as follows.

```

1 >>> from graphs import Q_Coloring
2 >>> qc = Q_Coloring(g)
3   Running a Gibbs Sampler for 42 step !
4   The q-coloring with 3 colors is feasible !!
5 >>> qc.showConfiguration()
6     v5 lightblue
7     v3 gold
8     v7 gold
9     v2 lightblue
10    v4 lightcoral
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

```

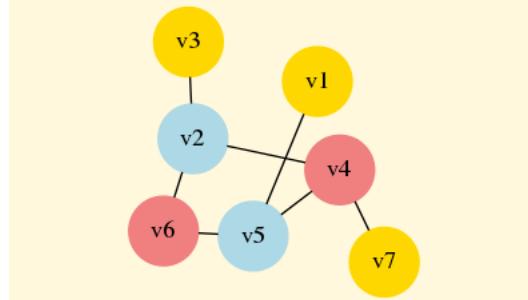


Fig. 22.2 3-Coloring of the tutorial graph

Graphs Python module (graphviz), R. Bisдорff, 2014

Actually, with the given tutorial graph instance, a 2-coloring is already feasible.

```

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

```

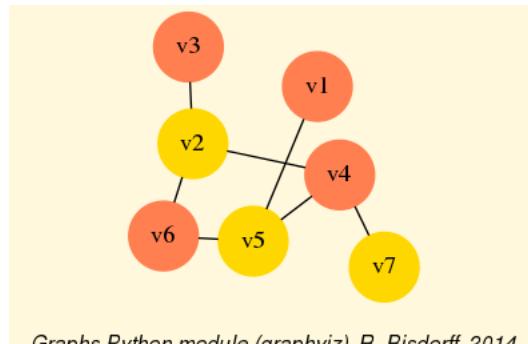


Fig. 22.3 3-Coloring of the tutorial graph

Graphs Python module (graphviz), R. Bischoff, 2014

22.3 MIS and clique enumeration

2-colorings define independent sets of vertices that are maximal in cardinality. Computing such MISs in a given Graph instance may be achieved by the showMIS method.

```

1 >>> g = Graph('tutorialGraph')
2 >>> g.showMIS()
3     *--- Maximal Independent Sets ---*
4     ['v2', 'v5', 'v7']
5     ['v3', 'v5', 'v7']
6     ['v1', 'v2', 'v7']
7     ['v1', 'v3', 'v6', 'v7']
8     ['v1', 'v3', 'v4', 'v6']
9     number of solutions:  5
10    cardinality distribution
11    card.: [0, 1, 2, 3, 4, 5, 6, 7]
12    freq.: [0, 0, 0, 3, 2, 0, 0, 0]
13    execution time: 0.00032 sec.
14    Results in self.misset
15 >>> g.misset

```

```

16     [frozenset({'v7', 'v2', 'v5'}),
17      frozenset({'v3', 'v7', 'v5'}),
18      frozenset({'v1', 'v2', 'v7'}),
19      frozenset({'v1', 'v6', 'v7', 'v3'}),
20      frozenset({'v1', 'v6', 'v4', 'v3'})]

```

A MIS in the dual of a graph instance g (its dual $-g$ Footnote[14]), corresponds to a maximal *clique*, i.e. a maximal complete subgraph in g . Maximal cliques may be directly enumerated with the `showCliques()` method.

```

1 >>> g.showCliques()
2     *--- Maximal Cliques ---*
3     ['v2', 'v3']
4     ['v4', 'v7']
5     ['v2', 'v4']
6     ['v4', 'v5']
7     ['v1', 'v5']
8     ['v2', 'v6']
9     ['v5', 'v6']
10    number of solutions:  7
11    cardinality distribution
12    card.: [0, 1, 2, 3, 4, 5, 6, 7]
13    freq.: [0, 0, 7, 0, 0, 0, 0, 0]
14    execution time: 0.00049 sec.
15    Results in self.cliques
16 >>> g.cliques
17     [frozenset({'v2', 'v3'}), frozenset({'v4', 'v7'}),
18      frozenset({'v2', 'v4'}), frozenset({'v4', 'v5'}),
19      frozenset({'v1', 'v5'}), frozenset({'v6', 'v2'}),
20      frozenset({'v6', 'v5'})]

```

22.4 Line graphs and maximal matchings

The `graphs` module also provides a `LineGraph` constructor. A *line graph* represents the adjacencies between edges of the given graph instance. We may compute for instance 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 become eventually an empty graph.

Notice that the MISs in the line graph provide *maximal matchings* –maximal sets of independent edges– of the original graph.

```

1 >>> c8 = CycleGraph(order=8)
2 >>> lc8 = LineGraph(c8)
3 >>> lc8.showMIS()
4 *--- Maximal Independent Sets ---*
5     [frozenset({'v3', 'v4'}), frozenset({'v5', 'v6'}), frozenset({'v1', 'v8'})]
6     [frozenset({'v2', 'v3'}), frozenset({'v5', 'v6'}), frozenset({'v1', 'v8'})]
7     [frozenset({'v8', 'v7'}), frozenset({'v2', 'v3'}), frozenset({'v5', 'v6'})]
8     [frozenset({'v8', 'v7'}), frozenset({'v2', 'v3'}), frozenset({'v4', 'v5'})]
9      [frozenset({'v7', 'v6'}), frozenset({'v3', 'v4'}), frozenset({'v1', 'v8'})]
10    [frozenset({'v2', 'v1'}), frozenset({'v8', 'v7'}), frozenset({'v4', 'v5'})]
11    [frozenset({'v2', 'v1'}), frozenset({'v7', 'v6'}), frozenset({'v4', 'v5'})]
12    [frozenset({'v2', 'v1'}), frozenset({'v7', 'v6'}), frozenset({'v3', 'v4'})]
13    [frozenset({'v7', 'v6'}), frozenset({'v2', 'v3'}), frozenset({'v1', 'v8'})],
14      frozenset({'v4', 'v5'})]
15    [frozenset({'v2', 'v1'}), frozenset({'v8', 'v7'}), frozenset({'v3', 'v4'})],
16      frozenset({'v5', 'v6'})]
17 number of solutions: 10

```

```

18     cardinality distribution
19     card.: [0, 1, 2, 3, 4, 5, 6, 7, 8]
20     freq.: [0, 0, 0, 8, 2, 0, 0, 0, 0]
21     execution time: 0.00029 sec.

```

The two last MISs of cardinality 4 (see Lines 13-16 above) give isomorphic perfect maximum matchings of the 8-cycle graph. Every vertex of the cycle is adjacent to a matching edge. Odd cycle graphs do not admit any perfect matching.

```

1 >>> maxMatching = c8.computeMaximumMatching()
2 >>> c8.exportGraphViz(fileName='maxMatchingcycleGraph', \
3 ...                                matching=maxMatching)
4 *---- exporting a dot file for GraphViz tools ----*
5 Exporting to maxMatchingcycleGraph.dot
6 Matching: {frozenset({'v1', 'v2'}),
7             frozenset({'v5', 'v6'}),
8             frozenset({'v3', 'v4'}),
9             frozenset({'v7', 'v8'}) }
10 circo -Tpng maxMatchingcycleGraph.dot\
11                               -o maxMatchingcycleGraph.png

```

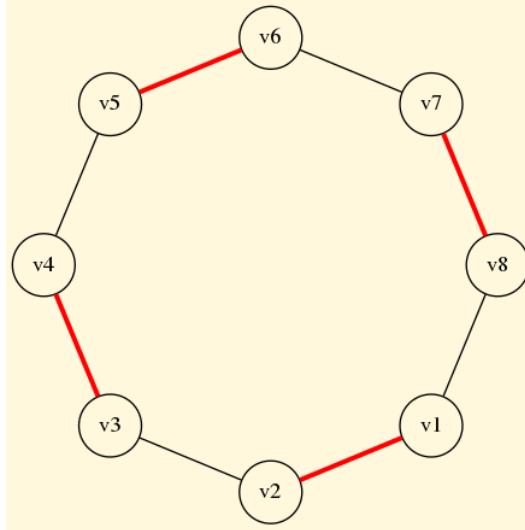


Fig. 22.4 A perfect maximum matching of the 8-cycle graph

Graphs Python module (graphviz), R. Bischoff, 2015

22.5 Grids and the Ising model

Special classes of graphs, like $n \times m$ rectangular or triangular grids (`GridGraph` and `IsingModel`) are available from the `graphs` module. For instance, we may use a *Gibbs* sampler again for simulating an *Ising Model* on such a grid.

```

1 >>> from graphs import GridGraph, IsingModel
2 >>> g = GridGraph(n=15, m=15)
3 >>> g.showShort()
4     ----- show short -----
5     Grid graph      : grid-6-6
6     n              : 6
7     m              : 6
8     order          : 36
9 >>> im = IsingModel(g, beta=0.3, nSim=100000, Debug=False)
10    Running a Gibbs Sampler for 100000 step !
11 >>> im.exportGraphViz(colors=['lightblue','lightcoral'])
12    ----- exporting a dot file for GraphViz tools -----
13    Exporting to grid-15-15-ising.dot
14    fdp -Tpng grid-15-15-ising.dot -o grid-15-15-ising.png

```

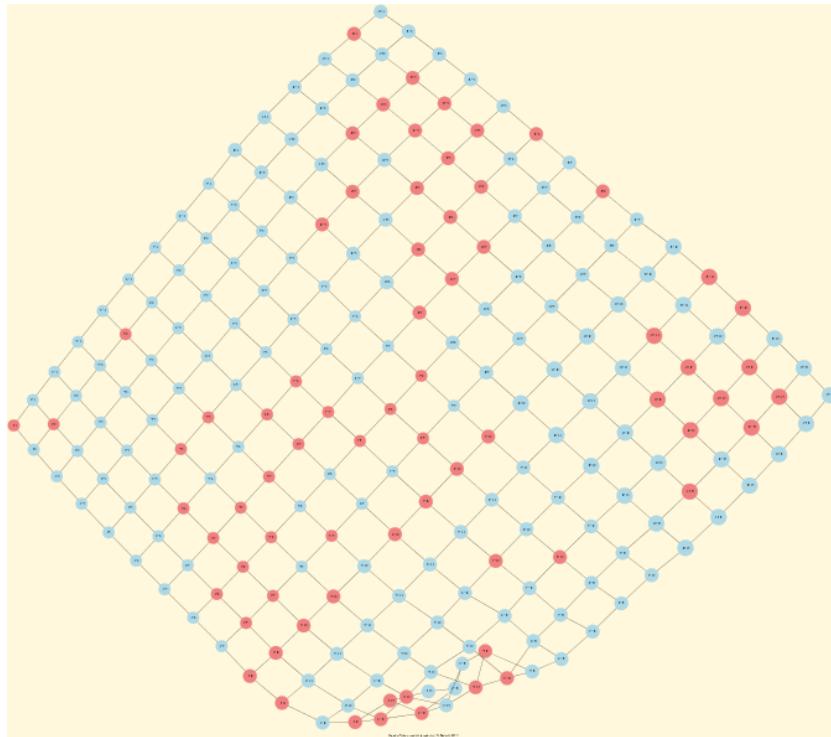


Fig. 22.5 Ising model of the 15×15 grid graph

22.6 Simulating Metropolis random walks

Finally, we provide the `MetropolisChain` class, a specialization of the `Graph` class, for implementing a generic *Metropolis* Monte Carlo Markov Chain (MCMC) sampler for simulating random walks on a given graph g :

```

1 >>> from graphs import Graph
2 >>> g = Graph(numberOfVertices=5, edgeProbability=0.5)
3 >>> g.showShort()
4     ----- short description of the graph -----
5     Name           : 'randomGraph'
6     Vertices       : ['v1', 'v2', 'v3', 'v4', 'v5']
7     Valuation domain : {'max': 1, 'med': 0, 'min': -1}
8     Gamma function :
9         v1 -> ['v2', 'v3', 'v4']
10        v2 -> ['v1', 'v4']
11        v3 -> ['v5', 'v1']
12        v4 -> ['v2', 'v5', 'v1']
13        v5 -> ['v3', 'v4']

```

following a given probability $probs = \{v1: x, v2: y, \dots\}$ for visiting each vertex:

```

1 >>> probs = {} # initialize a potential stationary probability
2             vector
3 >>> n = g.order # for instance: probs[v_i] = n-i/Sum(1:n) for i
4             in 1:n
5 >>> i = 0
6 >>> verticesList = [x for x in g.vertices]
7 >>> verticesList.sort()
8 >>> for v in verticesList:
9 ...     probs[v] = (n - i) / (n * (n+1) / 2)
10 ...     i += 1

```

The `checkSampling()` method of the the `MetropolisChain` class (see below) generates a random walk of $nSim = 30000$ steps on the given graph and records by the way the observed relative frequency with which each vertex is passed by:

```

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

```

In this example, the stationary transition probability distribution (see Listing above), shown by the `showTransitionMatrix()` method below is quite adequately simulated.

```

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
```

For the reader interested in algorithmic applications of Markov Chains we may recommend consulting O. Häggström's 2002 book: [FMCAA].

Chapter 23

On tree graphs and graph forests

Abstract

23.1 Generating random tree graphs

Using the `RandomTree` class from the `graphs` module, we may, for instance, generate a random tree graph with 9 vertices.

```
1 >>> from graphs import RandomTree
2 >>> t = RandomTree(order=9, seed=100)
3 >>> t
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 >>> t.exportGraphViz('tutRandomTree')
17     ----- exporting a dot file for GraphViz tools ---
18     Exporting to tutRandomTree.dot
19     neato -Tpng tutRandomTree.dot -o tutRandomTree.png
```

A tree graph of order n contains $n - 1$ edges (see Line 7 and 8) and we 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.

The structure of a tree graph of order $n > 2$ is entirely characterised by a corresponding PRÜFER *code*, i.e. a list of vertices keys- of length $n - 2$. See, for instance

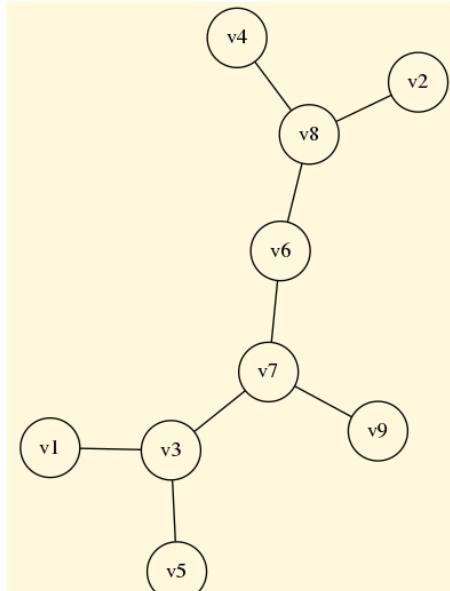


Fig. 23.1 Random tree graph instance of order 9

Graphs Python module (graphviz), R. Bischoff, 2019

in Line 15 the code `['v3', 'v8', 'v8', 'v3', 'v7', 'v6', 'v7']` corresponding to our sample tree graph t .

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', 'v7' becomes now 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 (see [BAR-1991]).

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 [BAR-1991]).

```

1 >>> verticesList = ['v1', 'v2', 'v3', 'v4', 'v5', 'v6', 'v7']
2 >>> n = len(verticesList)
3 >>> from random import seed, choice
4 >>> seed(101)
5 >>> code = []
6 >>> for k in range(n-2):
7     ...     code.append( choice(verticesList) )
8 >>> print(code)
9     ['v5', 'v7', 'v2', 'v5', 'v3']
10 >>> t = RandomTree(prueferCode=['v5', 'v7', 'v2', 'v5', 'v3'])
11 >>> t
  
```

```

12 *----- Graph instance description -----*
13 Instance class   : RandomTree
14 Instance name    : randomTree
15 Graph Order      : 7
16 Graph Size       : 6
17 Valuation domain : [-1.00; 1.00]
18 Attributes : ['name', 'order', 'vertices',
19             'valuationDomain', 'edges',
20             'prueferCode', 'size', 'gamma']
21 *---- RandomTree specific data ----*
22 Pruefer code   : ['v5', 'v7', 'v2', 'v5', 'v3']
23 >>> t.exportGraphViz('tutPruefTree')
24 *---- exporting a dot file for GraphViz tools -----
25 Exporting to tutPruefTree.dot
26 neato -Tpng tutPruefTree.dot -o tutPruefTree.png

```

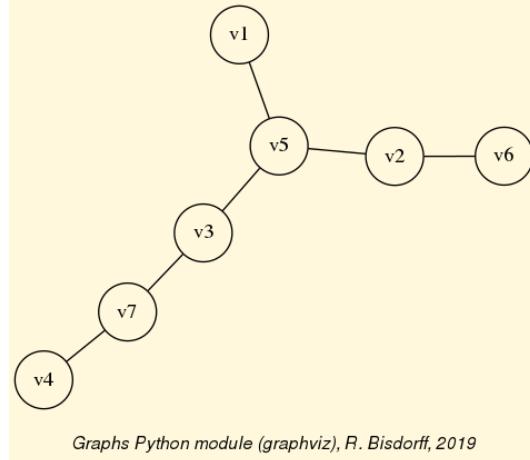


Fig. 23.2 Tree instance generated with a random PRÜFER code ['v5', 'v7', 'v2', 'v5', 'v3'].

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 recognize that it is in fact a tree instance ?

23.2 Recognizing tree graphs

Given a graph g of order n and size s , the following 5 assertions A1, A2, A3, A4 and A5 are all equivalent (see [BAR-1991]):

- A1 g is a tree;
- A2 g is without (chordless) cycles and $n = s + 1$;
- A3 g is connected and $n = s + 1$;
- A4Any two vertices of g^* are always connected by a unique path;

$A5g$ is connected and dropping any single edge will always disconnect g .

Assertion A3, for instance, gives a simple test for recognizing a tree graph. In case of a *lazy evaluation* of the test in Line 3 below, it is opportune, from a computational complexity perspective, to first, check the order and size of the graph, before checking its potential connectedness.

```

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

```

The random graph of order 8 and edge probability 30%, generated with seed 62, is actually a tree graph instance, as we may readily confirm from its graphviz drawing in Fig. 23.3.

```

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

```

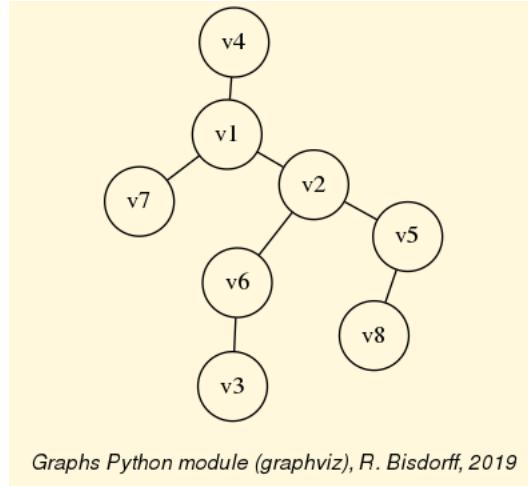


Fig. 23.3 Recognizing a tree instance. We may notice that vertex 'v2' is actually situated in the *centre* of the tree with a neighborhood depth of 2

Yet, we still have to recover its corresponding PRÜFER code. Therefore, we may use the `tree2Pruefer()` method.

```

1 >>> from graphs import TreeGraph
2 >>> g.__class__ = TreeGraph
3 >>> g.tree2Pruefer()
4 ['v6', 'v1', 'v2', 'v1', 'v2', 'v5']

```

In Fig. 23.3, we also notice that vertex 'v2' is actually situated in the *centre* of the tree with a neighborhood depth of 2. We may draw a correspondingly rooted and oriented tree graph.

```

1 >>> g.computeGraphCentres()
2     {'v2': 2}
3 >>> g.exportOrientedTreeGraphViz(\n4 ...     fileName='rootedTree', root='v2')\n5     ----- exporting a dot file for GraphViz tools\n6     Exporting to rootedTree.dot\n7     dot -Grankdir=TB -Tpng rootedTree.dot -o rootedTree.png

```

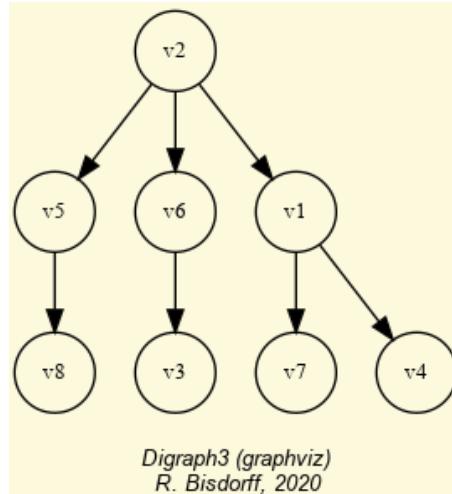


Fig. 23.4 Drawing an oriented tree rooted at its centre.

Digraph3 (graphviz)
R. Bisdorff, 2020

Let us now turn our attention toward a major application of tree graphs, namely *spanning trees* and *forests* related to graph traversals.

23.3 Spanning trees and forests

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 [WIL-1996]¹

```

1 >>> from graphs import RandomGraph,\n2 ...             RandomSpanningTree\n3 >>> g = RandomGraph(order=9,\n4 ...             edgeProbability=0.4, seed=100)\n5 >>> spt = RandomSpanningTree(g)

```

¹ WILSON's algorithm only works for connected graphs.

```

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

```

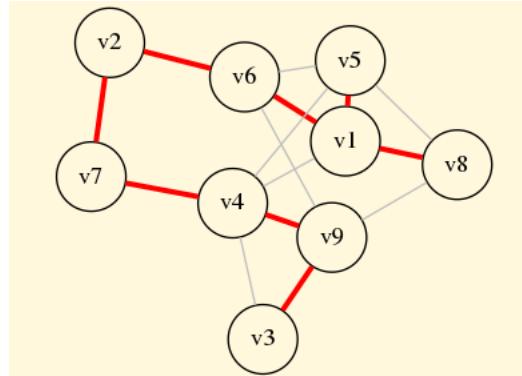


Fig. 23.5 Random spanning tree

Graphs Python module (graphviz), R. Bisendorff, 2019

More general, and in case of a not connected graph, we may generate with the *RandomSpanningForest* class a not necessarily uniform random instance of a *spanning forest* –one or more random tree graphs– generated from a random *depth first search* of the graph components’ traversals.

```

1 >>> g = RandomGraph(order=15, \
2 ...                           edgeProbability=0.1, seed=140)
3 >>> g.computeComponents()
4 [ {'v12', 'v01', 'v13'}, {'v02', 'v06'},
5   {'v08', 'v03', 'v07'}, {'v15', 'v11', 'v10', 'v04', 'v05'},
6   {'v09', 'v14'} ]

```

```

7 >>> spf = RandomSpanningForest(g, seed=100)
8 >>> spf.exportGraphViz(fileName='spanningForest', \
9 ...                                     WithSpanningTree=True)
10 *----- exporting a dot file for GraphViz tools -----*
11 Exporting to spanningForest.dot
12 [ ['v03','v07','v08','v07','v03'],
13   ['v13','v12','v13','v01','v13'],
14   ['v02','v06','v02'],
15   ['v15','v11','v04','v11','v15',
16     'v10','v05','v10','v15'],
17   ['v09','v14','v09']]
18 neato -Tpng spanningForest.dot -o spanningForest.png

```

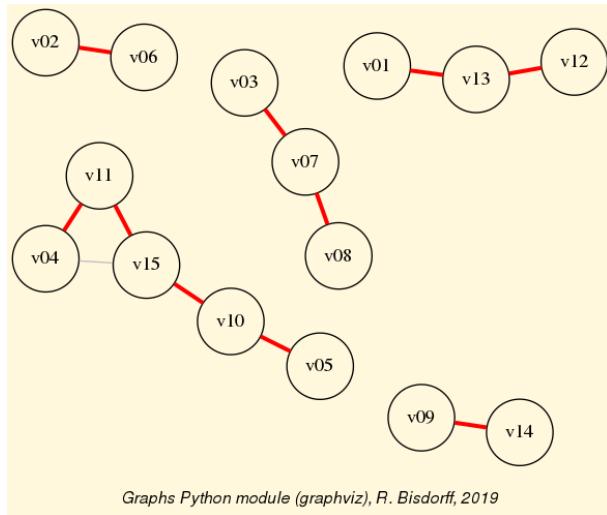


Fig. 23.6 Random spanning forest instance

23.4 Maximum determined spanning forests

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 Footnote[5] on the dual valuation of the graph [KRU-1956].

We consider, for instance, a randomly valued graph with five vertices and seven edges bipolar-valued in [-1.0; 1.0].

```

1 >>> from graphs import RandomValuationGraph
2 >>> g = RandomValuationGraph(seed=2)
3 >>> g

```

```

4 *----- Graph instance description -----*
5 Instance class      : RandomValuationGraph
6 Instance name       : randomGraph
7 Graph Order         : 5
8 Graph Size          : 7
9 Valuation domain   : [-1.00; 1.00]
10 Attributes         : ['name', 'order',
11                  'vertices', 'valuationDomain',
12                  'edges', 'size', 'gamma']

```

To inspect the edges' actual weights, we first transform the graph into a corresponding digraph (see Line 1 below) and use the `showRelationTable()` method (see Line 2 below) for printing its symmetric adjacency matrix.

Listing 23.1 Symmetric relation table

```

1 >>> dg = g.graph2Digraph()
2 >>> dg.showRelationTable()
3 *---- Relation Table ----
4   S | 'v1'  'v2'  'v3'  'v4'  'v5'
5   ---|-----
6   'v1' |  0.00  0.91  0.90 -0.89 -0.83
7   'v2' |  0.91  0.00  0.67  0.47  0.34
8   'v3' |  0.90  0.67  0.00 -0.38  0.21
9   'v4' | -0.89  0.47 -0.38  0.00  0.21
10  'v5' | -0.83  0.34  0.21  0.21  0.00
11  Valuation domain: [-1.00;1.00]

```

To compute the most determined spanning tree or forest, we may use the `BestDeterminedSpanningForest` constructor.

```

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 given graph 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 23.1 above).

```
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
```

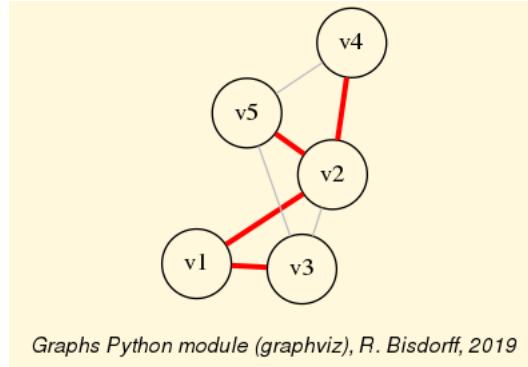


Fig. 23.7 Best determined spanning tree

Graphs Python module (graphviz), R. Bisдорff, 2019

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 24

Computing the non isomorphic MISs of the n-cycle graph

Abstract

Due to the public success of our common 2008 publication with Jean-Luc Marichal [ISOMIS-08] , we present in this chapter an example Python session for computing the non isomorphic maximal independent sets (MISs) from the 12-cycle graph, i.e. a CirculantDigraph class instance of order 12 and symmetric circulants 1 and -1 .

```
1 >>> from digraphs import CirculantDigraph
2 >>> c12 = CirculantDigraph(order=12,circulants=[1,-1])
3 >>> c12 # 12-cycle digraph instance
4     *----- Digraph instance description -----*
5     Instance class    : CirculantDigraph
6     Instance name     : c12
7     Digraph Order     : 12
8     Digraph Size      : 24
9     Valuation domain  : [-1.0, 1.0]
10    Determinateness   : 100.000
11    Attributes        : ['name', 'order', 'circulants',
12                          'actions', 'valuationdomain',
13                          'relation', 'gamma', 'notGamma']
```

Such n -cycle graphs are also provided as undirected graph instances by the CycleGraph class.

```
1 >>> from graphs import CycleGraph
2 >>> cg12 = CycleGraph(order=12)
3 >>> cg12
4     *----- Graph instance description -----*
5     Instance class    : CycleGraph
6     Instance name     : cycleGraph
7     Graph Order       : 12
8     Graph Size        : 12
9     Valuation domain  : [-1.0, 1.0]
10    Attributes        : ['name', 'order', 'vertices',
11                          'valuationDomain', 'edges',
12                          'size', 'gamma']
13 >>> cg12.exportGraphViz('cg12')
```

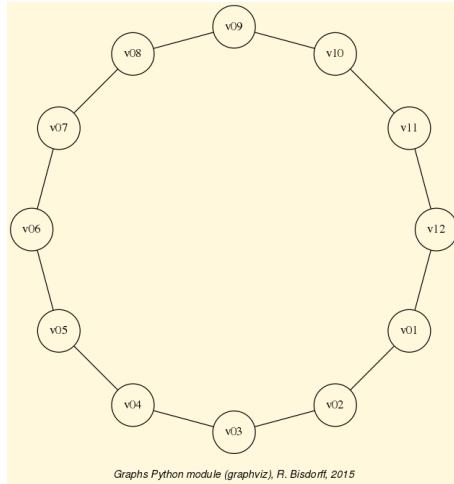


Fig. 24.1 The 12-cycle graph

24.1 Computing maximal independent sets (MISs)

A non isomorphic MIS corresponds in fact to a set of isomorphic MISs, i.e. an orbit of MISs under the automorphism group of the 12-cycle graph. We are now first computing all maximal independent sets that are detectable in the 12-cycle digraph with the `showMIS()` method.

```

1 >>> c12.showMIS(withListing=False)
2     *--- Maximal independent choices ---*
3     number of solutions:  29
4     cardinality distribution
5     card.:  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
6     freq.:  [0, 0, 0, 0, 3, 24, 2, 0, 0, 0, 0, 0, 0]
7     Results in c12.misset

```

In the 12-cycle graph, we observe 29 labelled MISs: 3 of cardinality 4, 24 of cardinality 5, and 2 of cardinality 6. In case of n-cycle graphs with $n \geq 20$, as the cardinality of the MISs becomes big, it is preferable to use the shell `perrinMIS` command compiled from C and installed Footnote[3] along with all the Digraphs3 python modules for computing the set of MISs observed in the graph.

```

1 ...$ echo 12 | /usr/local/bin/perrinMIS
2 # ----- #
3 # Generating MIS set of Cn with the      #
4 # Perrin sequence algorithm.           #
5 # Temporary files used.             #
6 # even versus odd order optimised.   #
7 # RB December 2006                   #
8 # Current revision Dec 2018          #
9 # ----- #
10 Input cycle order ? <-- 12

```

```

11 mis 1 : 100100100100
12 mis 2 : 010010010010
13 mis 3 : 001001001001
14 ...
15 ...
16 ...
17 mis 27 : 001001010101
18 mis 28 : 101010101010
19 mis 29 : 010101010101
20 Cardinalities:
21 0 : 0
22 1 : 0
23 2 : 0
24 3 : 0
25 4 : 3
26 5 : 24
27 6 : 2
28 7 : 0
29 8 : 0
30 9 : 0
31 10 : 0
32 11 : 0
33 12 : 0
34 Total: 29
35 execution time: 0 sec. and 2 millisec.

```

Reading in the result of the `perrinMIS` shell command, stored in a file called by default `curd.dat`, may be operated with the `readPerrinMisset()` method.

```

1 >>> c12.readPerrinMisset(file='curd.dat')
2 >>> c12.misset
3     {frozenset({'5', '7', '10', '1', '3'}),
4      frozenset({'9', '11', '5', '2', '7'}),
5      frozenset({'7', '2', '4', '10', '12'}),
6      ...
7      ...
8      ...
9      frozenset({'8', '4', '10', '1', '6'}),
10     frozenset({'11', '4', '1', '9', '6'}),
11     frozenset({'8', '2', '4', '10', '12', '6'})
12 }

```

24.2 Computing the automorphism group

For computing the corresponding non isomorphic MISs, we actually need the automorphism group of the `c12`-cycle graph. The `Digraph` class therefore provides the `automorphismGenerators()` method which adds automorphism group generators to a `Digraph` class instance with the help of the external shell `dreadnaut` command from the `nauty` software package Footnote[2].

```

1 >>> c12.automorphismGenerators()

```

```

2     ...
3     Permutations
4     {'1':'1', '2':'12', '3':'11', '4':'10',
5     '5':'9', '6':'8', '7':'7', '8':'6', '9':'5',
6     '10':'4', '11':'3', '12':'2'}
7     {'1':'2', '2':'1', '3':'12', '4':'11', '5':'10',
8     '6':'9', '7':'8', '8':'7', '9':'6', '10':'5',
9     '11':'4', '12':'3'}
10    >>> print('grpsize = ', c12.automorphismGroupSize)
11    grpsize = 24

```

The 12-cycle graph automorphism group is generated with both the permutations above and has group size 24.

24.3 Computing the isomorphic MISs

The `showOrbits()` method renders now the labelled representatives of each of the four orbits of isomorphic MISs observed in the 12-cycle graph (see Lines 7-10 below).

```

1 >>> c12.showOrbits(c12.misset,withListing=False)
2     ...
3     *---- Global result ----
4     Number of MIS: 29
5     Number of orbits : 4
6     Labelled representatives and cardinality:
7     1: ['2','4','6','8','10','12'], 2
8     2: ['2','5','8','11'], 3
9     3: ['2','4','6','9','11'], 12
10    4: ['1','4','7','9','11'], 12
11    Symmetry vector
12    stabilizer size: [1, 2, 3, ..., 8, 9, ..., 12, 13, ...]
13    frequency      : [0, 2, 0, ..., 1, 0, ..., 1, 0, ...]

```

The corresponding group stabilizers' 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 above), are illustrated in the corresponding unlabelled graphs of Fig. 24.1 below.

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'; the second orbit corresponds to writing four times a '3'. The third and fourth orbit correspond to writing two times a '3' and three times a '2'. There are two non isomorphic ways to do this latter circular sum. Either separating the '3's by one and two '2's, or by zero and three '2's (see Bisdorff & Marichal [ISOMIS-08]).

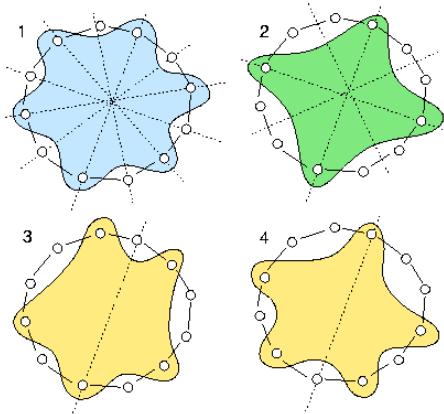


Fig. 24.2 The symmetry axes of the four non isomorphic MISs of the 12-cycle graph

Chapter 25

About split, interval and permutation graphs

Abstract

Contents

25.1 A multiply perfect graph

Following Martin Golumbic (see [GOL-2004] p. 149), we call a given graph g :

- *Comparability* graph when g is *transitively orientable*;
- *Triangulated* graph when g does not contain any *chordless cycle* of length 4 and more;
- *Interval* graph when g is *triangulated* and its dual $-g$ is a *comparability* graph;
- *Permutation* graph when g and its dual $-g$ bothe are *comparability* graphs;
- *Split* graph when g and its dual $-g$ bothe 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 *RandomIntervalIntersections-Graph* instance g (see Line 2 below).

```
1 >>> from graphs import\
2 ...           RandomIntervalIntersectionsGraph
3 >>> g = RandomIntervalIntersectionsGraph(\ \
4 ...                                         order=8, seed=100)
5 >>> g
6 *----- Graph instance description -----*
7 Instance class   : RandomIntervalIntersectionsGraph
8 Instance name    : randIntervalIntersections
9 Seed             : 100
10 Graph Order     : 8
11 Graph Size      : 23
12 Valuation domain : [-1.0; 1.0]
13 Attributes       : ['seed', 'name', 'order',
14   'intervals', 'vertices', 'valuationDomain',
15   'edges', 'size', 'gamma']
16 >>> print(g.intervals)
17 [(2, 7), (2, 7), (5, 6), (6, 8), (1, 8), (1, 1), (4, 7), (0, 10)]
```

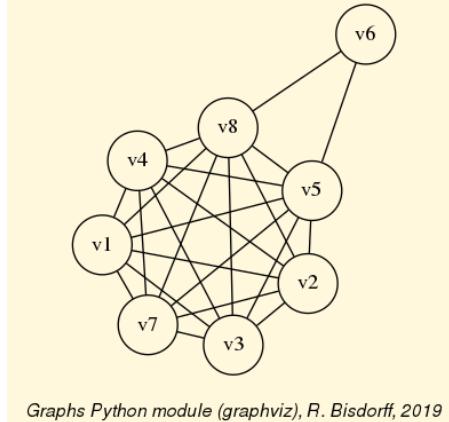
With $\text{seed} = 100$, we obtain here an interval graph, in fact apperfect graph, which is *conjointly* a triangulated, a comparability, a split and a permutation graph (see Listing 25.1 Lines 6, 10, 14).

Listing 25.1 Testing perfect graph categories.

```

1 >>> g.isPerfectGraph(Comments=True)
2   Graph 'randIntervalIntersections' is perfect !
3 >>> g.isIntervalGraph(Comments=True)
4   Graph 'randIntervalIntersections' is triangulated.
5   Graph 'dual_randIntervalIntersections' is transitively orientable.
6 => Graph 'randIntervalIntersections' is an interval graph.
7 >>> g.isSplitGraph(Comments=True)
8   Graph 'randIntervalIntersections' is triangulated.
9   Graph 'dual_randIntervalIntersections' is triangulated.
10 => Graph 'randIntervalIntersections' is a split graph.
11 >>> g.isPermutationGraph(Comments=True)
12   Graph 'randIntervalIntersections' is transitively orientable.
13   Graph 'dual_randIntervalIntersections' is transitively orientable.
14 => Graph 'randIntervalIntersections' is a permutation graph.
15 >>> print(g.computePermutation())
16   ['v5', 'v6', 'v4', 'v1', 'v3', 'v7', 'v8']
17   ['v8', 'v6', 'v1', 'v2', 'v3', 'v4', 'v7', 'v5']
18   [8, 2, 6, 5, 7, 4, 3, 1]
19 >>> g.exportGraphViz('randomSplitGraph')
20   ----- exporting a dot file for GraphViz tools -----
21   Exporting to randomSplitGraph.dot
22   fdp -Tpng randomSplitGraph.dot -o randomSplitGraph.png

```

**Fig. 25.1** A conjointly triangulated, comparability, interval, permutation and split graph*Graphs Python module (graphviz), R. Bisdorff, 2019*

In Fig. 25.1 we may readily recognize 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.

Notice however that the four properties:

1. *g* is a *comparability* graph;
2. *g* is a *co-comparability* graph, i.e. $-g$ is a comparability graph;
3. *g* is a *triangulated* graph;
4. *g* is a *co-triangulated* graph, i.e. $-g$ is a comparability graph;

are independent of one another (see [GOL-2004] p. 275).

25.2 Who is the liar?

Claude BERGE's famous mystery story (see [GOL-2004] p.20) may well illustrate the importance of being an *interval* graph.

Suppose the file `berge.py` Footnote[18] contains the following Graph instance data:

```

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 }
```

Six professors (labeled 'A', 'B', 'C', 'D', 'E' and 'I') had been to the library on the day that a rare tractate was stolen. Each entered once, stayed for some time, and then left. If two professors were in the library at the same time, then at least one of them saw the other. Detectives questioned the professors and gathered the testimonies that 'A' saw 'B' and 'E'; 'B' saw 'A' and 'I'; 'C' saw 'D' and 'I'; 'D' saw 'A' and 'I'; 'E' saw 'B' and 'I'; and 'I' saw 'C' and 'E'. This data is gathered in the previous file, where each positive edge $\{x,y\}$ models the testimony that, either x saw y , or y saw x .

```

1 >>> from graphs import Graph
2 >>> g = Graph('berge')
3 >>> g.showShort()
4     ----- short description of the graph -----
5     Name          : 'berge'
6     Vertices      : ['A', 'B', 'C', 'D', 'E', 'I']
7     Valuation domain : {'min': -1, 'med': 0, 'max': 1}
8     Gamma function :
9     A -> ['D', 'B', 'E']
```

```

10     B -> ['E', 'I', 'A']
11     C -> ['E', 'D', 'I']
12     D -> ['C', 'I', 'A']
13     E -> ['C', 'B', 'I', 'A']
14     I -> ['C', 'E', 'B', 'D']
15 >>> g.exportGraphViz('bergel')
16     *---- exporting a dot file for GraphViz tools -----*
17     Exporting to berge1.dot
18     fdp -Tpng berge1.dot -o berge1.png

```

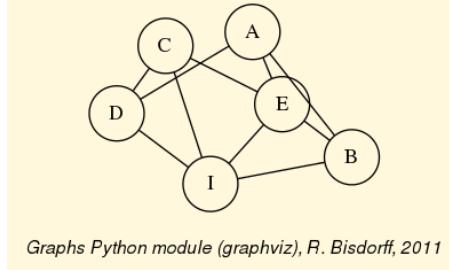


Fig. 25.2 Graph representation of the testimonies of the professors

Graphs Python module (graphviz), R. Bisdorff, 2011

From graph theory we know that time interval intersections graphs must in fact be interval graphs, i.e. *triangulated* and *co-comparative* graphs. The testimonies graph should therefore not contain any chordless cycle of four and more vertices. Now, the presence or not of such chordless cycles in the testimonies graph may be checked as follows:

```

1 >>> g.computeChordlessCycles()
2 Chordless cycle certificate: ['D', 'C', 'E', 'A', 'D']
3 Chordless cycle certificate: ['D', 'I', 'E', 'A', 'D']
4 Chordless cycle certificate: ['D', 'I', 'B', 'A', 'D']
5 [(['D', 'C', 'E', 'A', 'D'], frozenset({'C', 'D', 'E', 'A'})),
6 ([('D', 'I', 'E', 'A', 'D'), frozenset({'D', 'E', 'I', 'A'})),
7 ([('D', 'I', 'B', 'A', 'D'), frozenset({'D', 'B', 'I', 'A'})])

```

We see in the Listing above three intersection cycles of length 4, which is impossible to occur on the linear time line. Obviously one professor lied!

And it is *D*; if we put to doubt his testimony that he saw '*A*' (see Line 1 below), we obtain indeed a triangulated graph instance whose dual is a comparability graph.

```

1 >>> g.setEdgeValue( ('D', 'A'), 0)
2 >>> g.showShort()
3 *---- short description of the graph ----*
4     Name          : 'berge'
5     Vertices      : ['A', 'B', 'C', 'D', 'E', 'I']
6     Valuation domain : {'med': 0, 'min': -1, 'max': 1}
7     Gamma function :
8         A -> ['B', 'E']
9         B -> ['A', 'I', 'E']
10        C -> ['I', 'E', 'D']
11        D -> ['I', 'C']

```

```

12     E -> ['A', 'I', 'B', 'C']
13     I -> ['B', 'E', 'D', 'C']
14 >>> q.isIntervalGraph(Comments=True)
15     Graph 'berge' is triangulated.
16     Graph 'dual_berge' is transitively orientable.
17     => Graph 'berge' is an interval graph.
18 >>> g.exportGraphViz('berge2')
19     *---- exporting a dot file for GraphViz tools -----
20     Exporting to berge2.dot
21     fdp -Tpng berge2.dot -o berge2.png

```

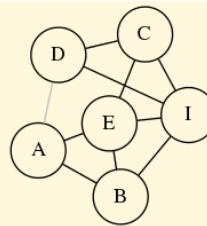


Fig. 25.3 The triangulated testimonies graph

Graphs Python module (graphviz), R. Bisendorff, 2011

25.3 Generating permutation graphs

A graph is called a *permutation* or *inversion* graph if there exists a permutation of its list of vertices such that the graph is isomorphic to the inversions operated by the permutation in this list (see [GOL-2004] Chapter 7, pp 157-170). This kind is also part of the class of perfect graphs.

```

1 >>> from graphs import PermutationGraph
2 >>> g = PermutationGraph(permuation = [4,3,6,1,5,2])
3 >>> g
4     *----- Graph instance description -----
5     Instance class    : PermutationGraph
6     Instance name     : permutationGraph
7     Graph Order       : 6
8     Permutation       : [4, 3, 6, 1, 5, 2]
9     Graph Size        : 9
10    Valuation domain : [-1.00; 1.00]
11    Attributes        : ['name', 'vertices', 'order',
12                          'permuation', 'valuationDomain',
13                          'edges', 'size', 'gamma']
14 >>> g.isPerfectGraph()
15     True
16 >>> g.exportGraphViz()
17     *---- exporting a dot file for GraphViz tools -----
18     Exporting to permutationGraph.dot

```

```

19   fdp -Tpng permutationGraph.dot \
20       -o permutationGraph.png

```

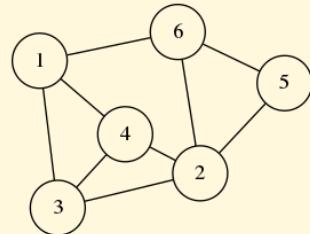


Fig. 25.4 The default Di-
GRAPH3 permutation graph

Graphs Python module (graphviz), R. Bisдорff, 2015

By using color sorting queues, the minimal vertex coloring for a permutation graph is computable in $O(n \log(n))$ (see [GOL-2004]).

```

1 >>> g.computeMinimalVertexColoring(Comments=True)
2     vertex 1: lightcoral
3     vertex 2: lightcoral
4     vertex 3: lightblue
5     vertex 4: gold
6     vertex 5: lightblue
7     vertex 6: gold
8 >>> g.exportGraphViz(fileName='coloredPermutationGraph', \
9                     WithVertexColoring=True)
9 ...
10    *---- exporting a dot file for GraphViz tools -----*
11    Exporting to coloredPermutationGraph.dot
12    fdp -Tpng coloredPermutationGraph.dot \
13        -o coloredPermutationGraph.png

```

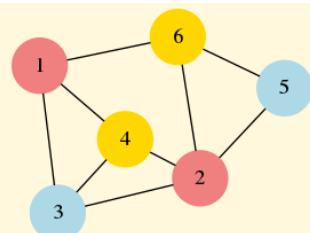


Fig. 25.5 Minimal vertex
coloring of the permutation
graph.

Graphs Python module (graphviz), R. Bisдорff, 2019

The correspondingly colored *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 (see Fig. 25.6).

```

1 >>> g.exportPermutationGraphViz(\n
2     ...             WithEdgeColoring=True)\n3     *---- exporting a dot file for GraphViz tools ----*\n4     Exporting to perm_permutationGraph.dot\n5     neato -n -Tpng perm_permutationGraph.dot\\n\n6         -o perm_permutationGraph.png

```

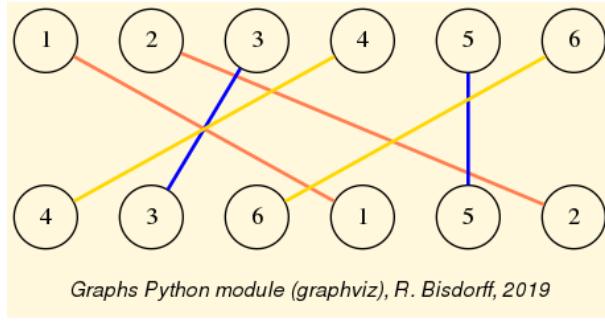


Fig. 25.6 Colored matching diagram of the permutation [4,3,6,1,5,2].

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 increasing alphabetic order of the adjacent vertices' keys (see [GOL-2004]). This orientation of the edges of a permutation graph is always transitive and delivers a *transitive ordering* of the vertices.

```

1 >>> dg = g.transitiveOrientation()\n2 >>> dg\n3     *----- Digraph instance description -----*\n4     Instance class    : TransitiveDigraph\n5     Instance name     : oriented_permutationGraph\n6     Digraph Order     : 6\n7     Digraph Size      : 9\n8     Valuation domain : [-1.00; 1.00]\n9     Determinateness   : 100.000\n10    Attributes        : ['name', 'order', 'actions',\n11                    'valuationdomain', 'relation',\n12                    'gamma', 'notGamma', 'size']\n13 >>> print('Transitivity degree: %.3f' %\n14     ...           dg.computeTransitivityDegree() )\n15     Transitivity degree: 1.000\n16 >>> dg.exportGraphViz(fileName='orientedPermGraph')\n17     *---- exporting a dot file for GraphViz tools ----*\n18     Exporting to orientedPermGraph.dot\n19     dot -Grankdir=TB -Tpng orientedPermGraph.dot\\n\n20         -o orientedPermGraph.png

```

The dual of a permutation graph is again a permutation graph and as such also transitively orientable.

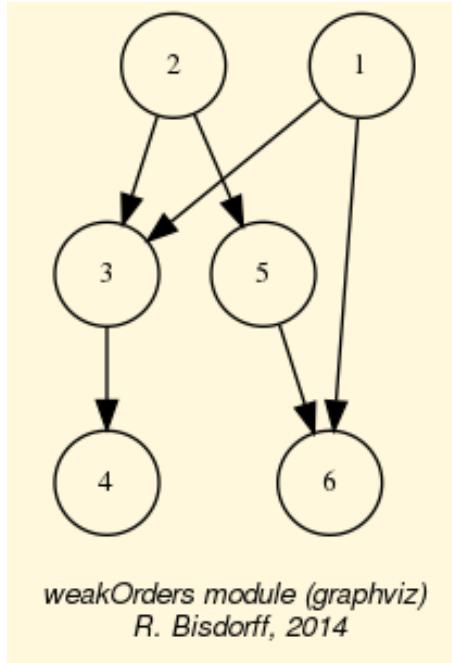


Fig. 25.7 The transitive orientation of the permutation graph.

```

1 >>> dgd = (-g).transitiveOrientation()
2 >>> print('Dual transitivity degree: %.3f' %\
3     ...           dgd.computeTransitivityDegree())
4     Dual transitivity degree: 1.00

```

25.4 Recognizing 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 recognizing 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,\n3     ...           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]
13 Attributes       : ['name', 'order', 'vertices',
14                      'valuationDomain', 'seed',
15                      'edges', 'size', 'gamma',
16                      'edgeProbability']
17 >>> g.isPerfectGraph()
18     True
19 >>> g.exportGraphViz()

```

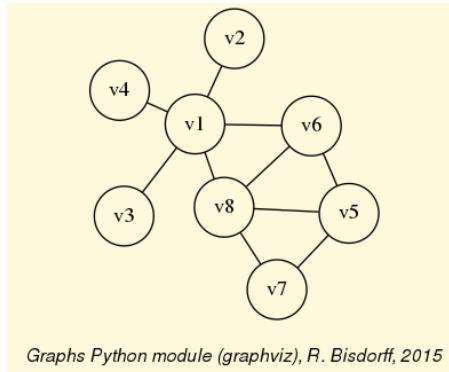


Fig. 25.8 Random graph of order 8 generated with edge probability 0.4.

Graphs Python module (graphviz), R. Bischoff, 2015

If the random perfect graph instance g , shown in Fig. 25.8, is indeed a permutation graph, g and its dual $-g$ should be *transitively orientable*, i.e. comparability graphs (see [GOL-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 a `edgeOrientations` attribute (see [GOL-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'): 1, ('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}

```

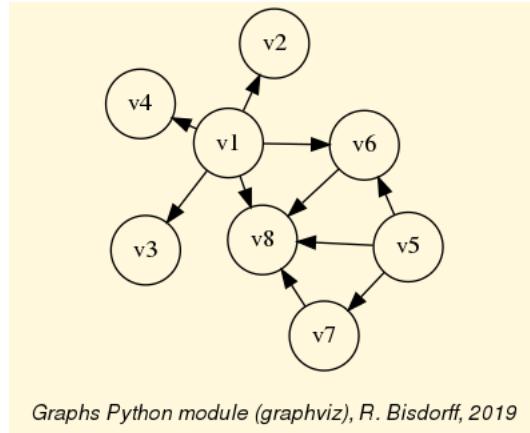


Fig. 25.9 Transitive neighbourhoods of the graph g .

Graphs Python module (graphviz), R. Bisendorff, 2019

The resulting orientation of the edges of g , shown in Fig. 25.9, is indeed transitive. The same procedure applied to the dual graph $gd = -g$ gives as well a transitive orientation to the edges of $-g$.

.. code-block:: pycon :linenos:

```

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}

```

Let us recheck these facts by explicitly constructing transitively oriented digraph instances with the `computeTransitivelyOrientedDigraph()` method ¹.

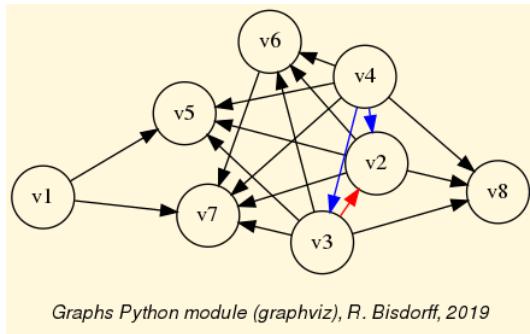
```

1 >>> og = g.computeTransitivelyOrientedDigraph(\_
2 ...     PartiallyDetermined = )
3 >>> print('Transitivity degree: %.3f' \%\
4 ...     (og.transitivityDegree))
5   Transitivity degree: 1.000
6 >>> ogd = (-g).computeTransitivelyOrientedDigraph(\_

```

¹ The `PartiallyDetermined = True*` flag (see Lines 2 and 7 above) is required here in order to orient only the actual edges of the graphs. Relations between vertices not linked by an edge are put to the indeterminate characteristic value 0. This allows us to compute, later on, convenient disjunctive digraph fusions.

Fig. 25.10 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.



```

7 ...                               PartiallyDetermined = True)
8 >>> print('Transitivity degree: %.3f' %\
9 ...           (ogd.transitivityDegree))
10 Transitivity degree: 1.000

```

As both graphs are indeed transitively orientable (see Lines 5 and 10 above), we may conclude that the given random graph g is actually a permutation graph instance. Yet, we still need to find now its corresponding permutation. We therefore implement a recipee given by Martin Golumbic [GOL-2004] p.159.

We will first *fuse* both og and ogd orientations above with an *epistemic disjunction* operated with the symmetric o-max operator Footnote[x].

```

1 >>> from digraphs import FusionDigraph
2 >>> f1 = FusionDigraph(og, ogd, operator='o-max')
3 >>> s1 = f1.computeCopelandRanking()
4 >>> print(s1)
5 ['v5','v7','v1','v6','v8','v4','v3','v2']

```

We obtain with the help of the `computeCopelandRanking()` method the linear ordering `['v5','v7','v1','v6','v8','v4','v3','v2']` of the vertices (see Line 5 above).

We reverse now the orientation of the edges in og (see $-og$ in Line 1 below) in order to generate, again by disjunctive fusion, the *inversions* that are produced by the permutation we are looking for.

Computing again a ranking with the COPELAND rule, will show the correspondingly permuted list of vertices (see Line 4 below).

```

1 >>> f2 = FusionDigraph((-og), ogd, operator='o-max')
2 >>> s2 = f2.computeCopelandRanking()
3 >>> print(s2)
4 ['v8','v7','v6','v5','v4','v3','v2','v1']

```

Vertex 'v8' is put from position 5 to position 1, vertex 'v7' is put from position 2 to position 1, vertex 'v6' from position 4 to position 2, vertex 'v5' from position 1 to position 3, etc We generate these position swaps for all vertices and obtain thus the required permutation (see Line 5 below).

```

1 >>> permutation = [0 for j in range(g.order)]
2 >>> for j in range(g.order):

```

```

3 ...     permutation[s2.index(s1[j])] = j+1
4 >>> print(permuation)
5 [5, 2, 4, 1, 6, 7, 8, 3]

```

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 (see [GOL-2004]). In our case here, we observe two different permutations and their reverses::

```

1 s1: ['v1', 'v4', 'v3', 'v2', 'v5', 'v6', 'v7', 'v8']
2 s2: ['v4', 'v3', 'v2', 'v8', 'v6', 'v1', 'v7', 'v5']
3 (s1 -> s2): [2, 3, 4, 8, 6, 1, 7, 5]
4 (s2 -> s1): [6, 1, 2, 3, 8, 5, 7, 4]

```

And

```

1 s3: ['v5', 'v7', 'v1', 'v6', 'v8', 'v4', 'v3', 'v2']
2 s4: ['v8', 'v7', 'v6', 'v5', 'v4', 'v3', 'v2', 'v1']
3 (s3 -> s4): [5, 2, 4, 1, 6, 7, 8, 3]
4 (s4 -> s3) = [4, 2, 8, 3, 1, 5, 6, 7]

```

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)
2 ['v1', 'v2', 'v3', 'v4', 'v5', 'v6', 'v7', 'v8']
3 ['v2', 'v3', 'v4', 'v8', 'v6', 'v1', 'v7', 'v5']
4 [2, 3, 4, 8, 6, 1, 7, 5]

```

We may finally check that, for instance, 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.

```

1 >>> gtesta = PermutationGraph(\
2 ...     permutation=[2, 3, 4, 8, 6, 1, 7, 5])
3 >>> gtestb = PermutationGraph(\
4 ...     permutations=[4, 2, 8, 3, 1, 5, 6, 7])
5 >>> gtesta.exportGraphViz('gtesta')
6 >>> gtestb.exportGraphViz('gtestb')

```

.. Figure:: isomorphicPerms.png :alt: Isomorphic permutation graphs :name: iso-morphicPermGraphs :width: 700 px :align: center

Isomorphic permutation graphs

And, we indeed recover in Fig. 25.11 indeed two isomorphic copies of the original random graph (compare with Fig. 25.8).

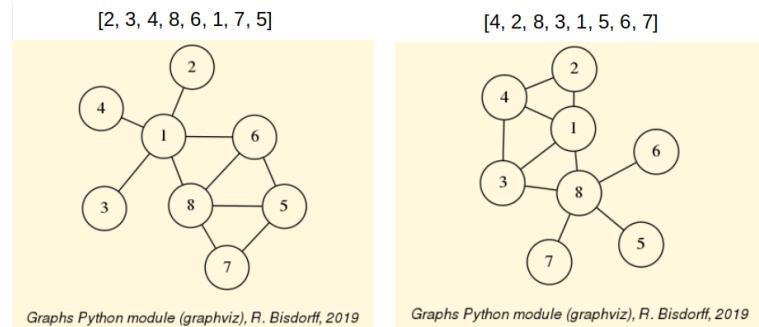


Fig. 25.11 Isomorphic permutation graphs.

