Generating random outranking digraphs

MICS: Algorithmic Decision Theory

Raymond Bisdorff

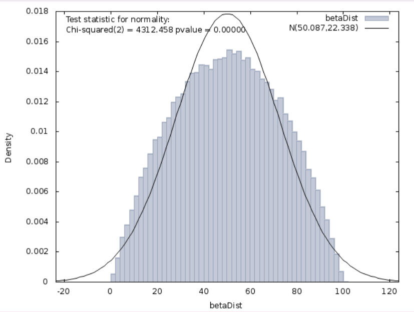
University of Luxembourg

April 28, 2020

Contents

Beta Performance Generator – 1

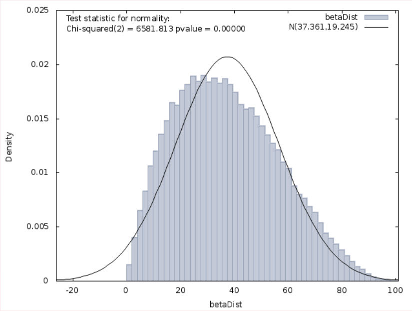
- The beta performance generator delivers random performance measures within a given performance scale following a $\mathcal{B}eta(\alpha, \beta)$ probability law.

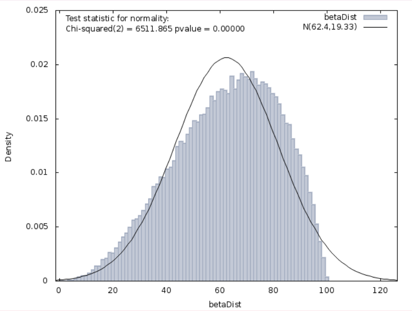


- In the default case, ($\alpha = 2.0, \beta = 2.0$), the mode *xm* is situated in the middle (50.0) of the performance scale $[0.0, 100.0]$ and the probabilty is equally distributed on both sides, i.e. *xm* represents the median performance, and the standard deviation $sd \approx 15.0$.

Beta Performance Generator – 2

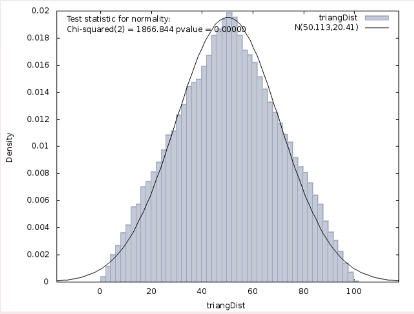
- We consider two variants with equal standard deviation $sd = 15$:
  - low performances: $xm = 25$ ($\alpha = 2.0$, $\beta = \frac{1.0}{1.0 - xm}$),
  - high performances: $xm = 75$ ($\alpha = \frac{1.0}{1.0 - xm}$, $\beta = 2.0$),





Triangular Performance Generator – 1

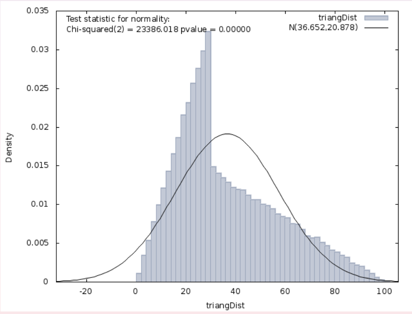
- The triangular performance generator delivers random performance measures within a given performance scale following an extended triangular alert$\mathcal{T}r(xm, r)$ probability law with mode xm and probability repartition r lower or equal xm.

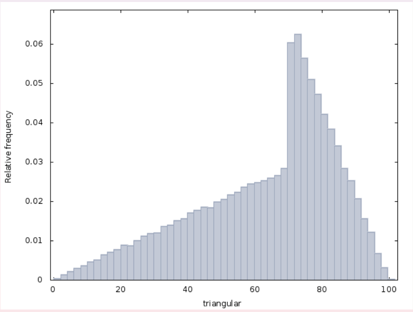


- In the default case, the mode xm is situated in the middle (50.0) of the performance scale and the probabilty is equally distributed on both sides, i.e. $r = 0.5$ and xm represents the median performance measure.

Triangular Performance Generator – 2

- We consider two variants with fixed repartition $r = 0.5$:
  - low performances: $xm = 30$,
  - high performances: $xm = 70$,

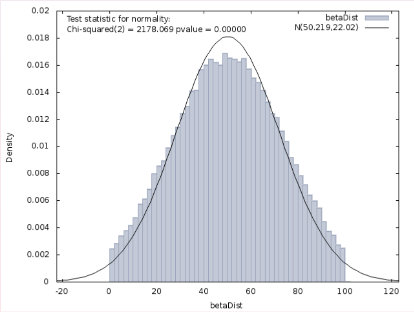




See the Digraph3 <RandomNumbers> module.

Truncated Gaussian Performance Generator – 1

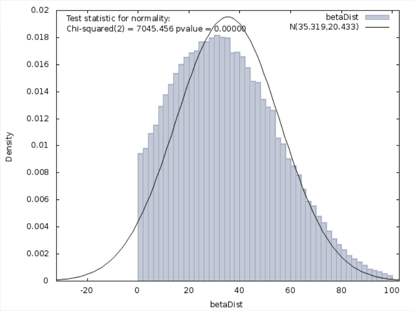
- The truncated Gaussian performance generator delivers random performance measures within a given performance scale following a truncated $\mathcal{N}(\mu, \sigma)$ probability law with mean μ and standard deviation σ.

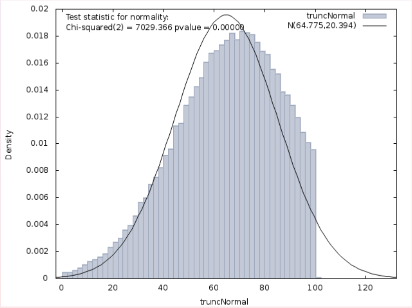


- In the default case, the mode xm is situated in the middle (50.0) of the performance scale and the standard deviation is a fourth (25.0) of the scale scope.

Truncated Gaussian Performance Generator – 2

- We consider two variants:
  - low performances: $xm = 30$,
  - high performances: $xm = 70$,





A Standard Random Performance Tableau

- 20 decision actions; low variant: 13; high variant: 50;
- 13 criteria; low variant: 7; high variant: 21;
- All criteria are by default equi-significant (same weight 1); uniform random weights may be generated within a givn weight scale;
- All criteria use a same cardinal performance measurement scale; from 0.0 to 100.0 by default: user provided scale limits may be given;
- Individual performances are by default generated with a beta law: $\mathcal{Beta}(2, 2)$. Two variants are provided:
  - a uniform law: $\mathcal{U}(a, b)$ with a and b the performance measurement scale limits;
  - an extended triangular law: $\mathcal{T}(xm, r)$, where xm is the mode and r the percentile of xm.

  See the Digraph3 RandomPerformanceTableau class description

Fixed Discrimination Thresholds

- On each criterion, the default discrimination thresholds are chosen in percentages of the amplitude of the criterion performance measurement scales:
  - indifference threshold equals 2.5% of the potential performance amplitude;
  - preference threshold equals 5.0% of the potential performance amplitude;
  - veto threshold equals 80.0% of the potential performance amplitude.
- Note: Ordinal criteria admit by default solely a preference threshold of one unit.

The Digraph3 <RandomPerformanceTableau> class

Example Python session:

```
>>> from randomPerfTabs import RandomPerformanceTableau
>>> t = RandomPerformanceTableau(numberOfActions=13,\
          numberOfCriteria=7,weightDistribution='random',\
          weightScale=(0.0,10.0),seed=100,\
          missingDataProbability=0.03)
>>> t
*------- PerformanceTableau instance description ------*
 Instance class: RandomPerformanceTableau
 Seed          : 100
 Instance name : randomperftab
 # Actions       : 13
 # Criteria      : 7
 Attributes    : ['weightPreorder','BigData','criteria',
              'missingDataProbability','commonScale',
              'evaluation','digits','name','sumWeights',
              'commonMode','randomSeed','actions']
>>> t.showHTMLPerformanceTableau(Transposed=True,\
                  title='Standard performance tableau')
```


Standard performance tableau

criterion	a01	a02	a03	a04	a05	a06	a07	a08	a09	a10	a11	a12	a13
g1	76.12	27.99	58.55	44.59	43.58	53.48	75.81	87.68	43.83	38.12	25.36	23.61	94.28
g2	53.54	34.36	31.88	31.00	44.85	48.34	46.76	34.34	NA	79.97	29.50	53.59	88.90
g3	70.84	95.04	30.74	83.81	56.50	63.09	42.17	51.07	33.63	54.63	24.58	50.33	7.41
g4	41.59	73.65	78.28	75.01	69.26	77.84	76.50	28.84	22.32	64.32	35.38	40.52	73.26
g5	NA	35.61	77.25	60.77	74.02	34.83	64.77	84.78	53.76	43.09	15.31	43.67	49.74
g6	16.39	26.25	NA	81.29	63.48	50.86	73.49	48.74	17.26	40.48	47.92	67.83	43.61
g7	46.69	61.65	7.12	72.87	20.62	65.10	67.77	64.11	3.46	73.21	52.46	69.45	67.01

On each criterion $g1$ to $g7$, the performances of the seven decision alternatives are generated on a common 0.00 to 100.00 satisfaction scale. The light green marked cells indicate the best performance obtained on this criterion, whereas the light red marked cells indicate the weakestt performance obtained on this criterion. On criterion $g1$, for instance, alternative $a12$ show the weakest and $a13$ the best performance.

Notice by the way the three missing evaluations: one for alternative $a1$ on criterion $g5$, one for $a03$ on criterion $g6$ and one for $a09$ on criterion $g2$.

<RandomPerformanceTableau> class

Example Python session –continue:

```
>>> t.showCriteria(IntegerWeights=True)
 *----  criteria -----*
 g1 'RandomPerformanceTableau() instance'
 Scale = (0.0, 100.0)
 Weight = 8
 Threshold ind : 2.50 + 0.00x ; percentile:  0.06
 Threshold pref : 5.00 + 0.00x ; percentile:  0.09
 Threshold veto : 80.00 + 0.00x ; percentile:  1.0
 ...
 ...
```

On criterion $g1$, 6% of the performance differences are insignificant, 9% are below the preference discrimination threshold, and no considerable performance difference is observed.

We may visualize a ranked heatmap of the performance tableau with the t.showHTMLPerformanceHeatmap method.

Example Python session –continue:

```
>>> t.showHTMLPerformanceHeatmap(Transposed=True,\
        pageTitle='Ranked heatmap of the decision alternati
        rankingRule='Copeland',colorLevels=5)
```

Ranked heatmap of the decision alternatives

criteria	weight	a07	a04	a08	a05	a06	a01	a13	a03	a12	a10	a02	a09	a11
g5	+7.00	64.77	60.77	84.78	74.02	34.83	NA	49.74	77.25	43.67	43.09	35.61	53.76	15.31
g6	+7.00	73.49	81.29	48.74	63.48	50.86	16.39	43.61	NA	67.83	40.48	26.25	17.26	47.92
g1	+6.00	75.81	44.59	87.68	43.58	53.48	76.12	94.28	58.55	23.61	38.12	27.99	43.83	25.36
g3	+6.00	42.17	83.81	51.07	56.50	63.09	70.84	7.41	30.74	50.33	54.63	95.04	33.63	24.58
g2	+5.00	46.76	31.00	34.34	44.85	48.34	53.54	88.90	31.88	53.59	79.97	34.36	NA	29.50
g4	+2.00	76.50	75.01	28.84	69.26	77.84	41.59	73.26	78.28	40.52	64.32	73.65	22.32	35.38
g7	+2.00	67.77	72.87	64.11	20.62	65.10	46.69	67.01	7.12	69.45	73.21	61.65	3.46	52.46

Color legend:

quantile	20.00%	40.00%	60.00%	80.00%	100.00%

The criteria appear ordered by decreasing significance weight, whereas the decision alternatives are ranked following the *Copeland* ranking rule. See the Digraph3 tutorial on ranking with multiple incommensurable criteria.

Random *Cost-Benefit* Performance Tableau – I

- 20 decision actions; low variant: 13; high variant: 50.
- 13 criteria; low variant: 7; high variant: 20.
- A criteria is with equal probability either to be minimized (cost criteria) or to be maximized (benefit criteria).
- All criteria either support an ordinal or a cardinal performance scale; the cost criteria being mostly cardinal (2/3) and the benefit ones mostly ordinal (2/3).
- Ordinal performances are represented on integer scales: $\{1, 2, ..., 10\}$.
- Cardinal performances are represented on a decimal scale: $[0.0; 100.0]$ with a precision of 2 digits.

Random Cost-Benefit Performance Tableau – II

- In the Cost-Benefit model the decision actions are divided randomly into three categories: *cheap*, *neutral*, *advantageous*.
- An action is called:
  - cheap when the performances are generated with $\mathcal{T}(xm = 30, r = 0.5)$ (default), $\mathcal{N}(\mu = 30, \sigma = 25)$, or $\mathcal{B}eta(\alpha = 2.62203, \beta = 5.8661)$, i.e. $(xm = 25, sd = 15)$.
  - advantageous when the performances are generated with $\mathcal{T}(xm = 70, r = 0.5)$ (default), $\mathcal{N}(\mu = 70, \sigma = 25)$, or $\mathcal{B}eta(\alpha = 5.8661, \beta = 2.62203)$, i.e. $(xm = 75, sd = 15)$
  - and neutral when the performances are generated with $\mathcal{T}(xm = 50, r = 0.5)$ (default), $\mathcal{N}(\mu = 50, \sigma = 25)$, or $\mathcal{B}eta(\alpha = 5.055, \beta = 5.055)$, i.e. $(xm = 50, sd = 15)$

Random *Cost-Benefit* Performance Tableau – II

Fixed Percentile Discrimination Thresholds:
On each cardinal criterion, the default performance discrimination thresholds are chosen such that the:

- indifference threshold equals the percentile 5 of all generated performance differences;
- preference threshold equals the percentile 10 of all generated performance differences;
- veto threshold equals the percentile 95 of all generated performance differences.

<RandomCBPerformanceTableau> class – I

```
>>> from randomPerfTabs import RandomCBPerformanceTableau
>>> t = RandomCBPerformanceTableau(numberOfActions=7,\
    numberOfCriteria=11,commonPercentiles=\
    {'ind':0.05, 'pref':0.10 , 'veto':0.95},\
    missingDataProbability=0.05,seed=109)
>>> t.showCriteria(IntegerWeights=True)
c1 'Costs/random cardinal cost criterion'
  Scale = (0.0, 100.0)
  Weight = 7
  Threshold ind : 7.64 + 0.00x ; percentile:  0.048
  Threshold pref : 8.11 + 0.00x ; percentile:  0.14
  Threshold veto : 62.25 + 0.00x ; percentile:  0.95
...
b2 'Benefits/random ordinal benefit criterion'
  Scale = (0, 10)
  Weight = 4
...
```

In this example we notice, for instance, a cardinal *Costs* criterion *c*1 of weight 7 with default performance discrimination thresholds and an ordinal *Benefits* criterion *b*2 of weight 4.

<RandomCBPerformanceTableau> class – II

Continue –

```
>>> t.showActions()
*----- show decision action --------------*
key:  a1
 short name: a1n
 name:       random neutral decision action
key:  a2
 short name: a2c
 name:       random cheap decision action
...
key:  a5
 short name: a5a
 name:       random advantageous decision action
...
>>> t.showHTMLPerformanceTableau(\
  title='Cost-Benefit Performance Tableau')
```


Cost-Benefit Performance Tableau

criteria	b01	b02	b03	b04	b05	b06	b07	c01	c02	c03	c04
weight	**4.00**	**4.00**	**4.00**	**4.00**	**4.00**	**4.00**	**4.00**	**7.00**	**7.00**	**7.00**	**7.00**
a1n	59.11	4.00	8.00	31.64	4.00	68.21	NA	-66.27	-58.09	-44.53	-62.89
a2c	28.32	3.00	5.00	57.22	5.00	46.25	29.96	-48.44	-55.27	-38.31	-27.74
a3n	46.03	9.00	5.00	61.17	5.00	84.36	61.91	-56.55	-59.05	-49.73	-71.00
a4n	27.69	8.00	5.00	55.57	7.00	75.17	22.04	-26.20	-15.65	-59.28	-47.26
a5a	NA	8.00	8.00	57.59	9.00	25.83	41.54	-77.72	-67.53	-83.83	NA
a6c	82.24	5.00	5.00	12.67	2.00	11.80	65.86	-89.75	-36.31	-27.42	-76.57
a7c	26.72	3.00	3.00	21.20	2.00	NA	39.16	-64.58	-43.29	-35.74	-49.99

The sum of weights of the *Benefits* criteria ($7 \times 4 = 28$) equals the sum of weights of the *Costs* criteria ($4 \times 7 = 28$). We observe 3 cheap actions (*a*2, *a*6, *a*7), three neutral actions (*a*1, *a*3, *a*4) and one advantageous action (*a*5). As costs must be mimized, the performances registered on the *Costs* criteria are all negative. Cheap actions *a*6 and *a*7 show five, respectively four weakest performances, whereas neutral action *a*3 shows three best performances.

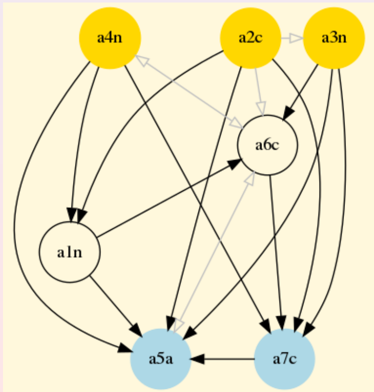
The random outranking digraph

```
>>> from outrankingDigraphs import BipolarOutrankingDigraph
>>> g = BipolarOutrankingDigraph(t,Normalized=True)
>>> g.showRelationTable()
>>> # strict(codual) outranking digraph drawing
>>> (~(-g)).exportGraphViz(bestChoice=['a2','a3','a4'],\
            worstChoice=['a5','a7'])
```

Normalized Relation

r(x S y)	a1	a2	a3	a4	a5	a6	a7
a1	–	-0.23	0.11	-0.25	0.30	0.14	0.11
a2	0.36	–	0.29	0.07	0.23	1.00	0.54
a3	0.27	0.00	–	0.11	0.52	1.00	0.18
a4	0.39	0.29	0.04	–	1.00	0.00	0.54
a5	-0.16	-0.09	-0.52	-1.00	–	0.00	-0.02
a6	-0.14	0.00	-1.00	0.00	0.00	–	1.00
a7	0.14	-0.14	-0.05	-0.29	0.16	-1.00	–

Valuation domain: [-1.00; +1.00]



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

Generating random public policies

- we consider three decision objectives: economical aspects, environmental aspects and societal aspects.
- Every performance criteria is affected randomly to one of the three objectives.
- The three objectives are equally important and the criteria in each objective are equally significant.
- Each random potential public policy is allocated on each objective to one of three performance categories: low performances ($-$), medium performances ($\sim$) or high performances ($+$).
- When generating now the performances of a policy on a criterion, the random generator is modulated following the performance profile of the policy respective to each decision objective.

<Random3ObjectivesPerformanceTableau> class

```
>>> from randomPerfTabs import \
        Random3ObjectivesPerformanceTableau
>>> t = Random3ObjectivesPerformanceTableau(\
  numberOfActions=7,numberOfCriteria=13,seed=100)
>>> t.showObjectives()
*------ show objectives -------*
Eco: Economical aspect
  ec01 criterion of objective Eco 24
  ec04 criterion of objective Eco 24
  ...
 Total weight: 72.00 (3 criteria)
Soc: Societal aspect
  so02 criterion of objective Soc 12
  so05 criterion of objective Soc 12
  ...
  ...
 Total weight: 72.00 (6 criteria)
 Env: Environmental aspect
  en03 criterion of objective Env 18
  en08 criterion of objective Env 18
  ...
 Total weight: 72.00 (4 criteria)
>>> ...
```

<Random3ObjectivesPerformanceTableau> class

Continue –

```
>>> t.showActions()
*----- show decision action --------------*
key:  p1
 short name:  p1
 name:        random decision action Eco+ Soc~ Env~
 profile:     {'Eco':'good', 'Soc':'fair', 'Env':'fair'}
key:  p2
 short name:  p2
 name:        random decision action Eco+ Soc- Env~
 profile:     {'Eco':'good', 'Soc':'weak', 'Env':'fair'}
...
key:  p6
 short name:  p6
 name:        random decision action Eco- Soc~ Env~
 profile:     {'Eco':'fair', 'Soc':'fair', 'Env':'good'}
...
>>> t.showHTMLPerformanceHeatmap(\
  pageTitle='Performance heatmap of random public policies',\
      colorLevels=5,Correlations=True)
```


Performance heatmap of random public policies

criteria	ec07	en13	ec04	ec01	so06	so02	en10	so11	en03	so09	en08	so12	so05
weights	+24.00	+18.00	+24.00	+24.00	+12.00	+12.00	+18.00	+12.00	+18.00	+12.00	+18.00	+12.00	+12.00
tau(*)	+0.48	+0.45	+0.40	+0.40	+0.38	+0.24	+0.21	+0.19	+0.14	+0.07	+0.00	-0.12	-0.12
p6	55.57	68.83	53.77	75.31	66.21	49.05	78.60	NA	88.48	50.92	44.62	47.54	49.69
p2	87.02	77.66	70.30	72.54	35.52	15.33	21.83	53.73	77.19	NA	70.32	77.56	36.31
p4	64.37	25.14	80.40	85.67	43.54	89.12	17.94	56.36	20.79	62.18	85.20	37.60	55.60
p5	36.15	54.60	86.59	NA	3.85	82.36	20.30	81.22	46.78	34.65	91.94	72.54	63.58
p1	72.59	37.14	29.25	37.78	59.35	35.11	52.12	67.40	34.71	49.39	18.86	87.32	82.68
p7	26.95	27.04	8.96	46.75	31.22	7.42	50.04	7.31	70.45	9.83	72.53	13.94	29.52
p3	28.41	41.37	9.71	16.33	19.02	37.39	9.17	41.98	70.11	60.27	61.70	77.81	51.00

Color legend:

quantile	20.00%	40.00%	60.00%	80.00%	100.00%

(*) tau: *Ordinal (Kendall) correlation between marginal criterion and global ranking relation*
Ranking rule: **NetFlows**
Ordinal (Kendall) correlation between global ranking and global outranking relation: **+0.815**

The performance criteria are ordered in decreasing marginal correlation with the default '*NetFlows*' ranking of the seven potential public policies. Overall best performing policy appears to be policy *p*6, follwed by *p*2. Weakest policy is *p*3. The three criteria, supporting the economic decision objective (*ec*07, *ec*04 and *ec*01), appear most correlated with the proposed ranking.

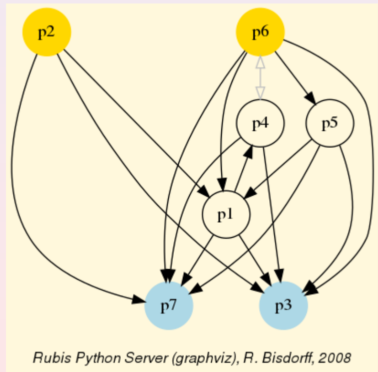
The random outranking digraph

```
>>> from outrankingDigraphs import BipolarOutrankingDigraph
>>> g = BipolarOutrankingDigraph(t,Normalized=True)
>>> g.showHTMLRelationTable()
>>> # strict (codual) outranking digraph drawing
>>> (~(-g)).exportGraphViz(bestChoice=['p2','a6'],\
                 worstChoice=['p3','p7'])
```

Normalized Relation

r(x S y)	p1	p2	p3	p4	p5	p6	p7
p1	–	-0.22	0.56	0.17	-0.06	-0.33	0.56
p2	0.28	–	0.72	0.06	0.11	0.22	1.00
p3	-0.22	-0.44	–	-1.00	-0.28	-1.00	0.33
p4	-0.06	0.28	1.00	–	0.08	0.00	0.67
p5	0.06	0.11	0.44	0.39	–	-0.11	0.44
p6	0.50	0.08	1.00	0.00	0.11	–	0.78
p7	-0.28	-1.00	0.28	-0.50	-0.33	-0.78	–

Valuation domain: [-1.00; +1.00]



Random academic performance tableaux – I

The `randomPerfTabs.RandomAcademicPerformanceTableau` class generates temporary performance tableaux with random grades for a given number of students in different courses.

Parameters:

- Number of students and 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).

Random academic performance tableaux – II

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 the default 0-20 grading range, the random grading 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 a *double triangular probablity 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 (see the documentation of `randomNumbers` module).

<RandomAcademicPerformanceTableau> class

```
>>> from randomPerfTabs import\
        RandomAcademicPerformanceTableau
>>> t = RandomAcademicPerformanceTableau(\
        numberOfStudents=13,numberOfCourses=7,\
        missingDataProbability=0.03,\
        WithTypes=True, seed=100)
>>> t
 *------- PerformanceTableau instance description ------*
 Instance class: RandomAcademicPerformanceTableau
 Seed          : 100
 Instance name : randstudPerf
 # Actions     : 13
 # Criteria    : 7
 Attributes    : ['randomSeed', 'name', 'actions',
       'criteria', 'evaluation', 'weightPreorder']
>>> t.showHTMLPerformanceHeatmap(Transposed=True,\
       colorLevels=5,ndigits=0)
```

Heatmap of Performance Tableau 'randstudPerf'

criteria	weight	s06	s08	s09	s05	s07	s04	s01	s12	s02	s03	s10	s11	s13
g6	+4.00	14	12	14	13	12	13	13	14	10	10	9	10	5
g2	+3.00	17	16	14	14	14	13	14	11	13	10	13	10	10
g5	+3.00	13	13	12	NA	12	13	9	10	12	10	NA	10	NA
g1	+2.00	16	14	15	13	NA	13	13	13	10	16	11	10	10
g3	+2.00	16	14	12	12	12	11	12	11	12	15	13	10	10
g4	+1.00	14	14	11	19	14	13	13	13	16	13	NA	10	10
g7	+1.00	14	15	9	17	12	11	12	12	18	13	11	10	10

Color legend:

quantile	20.00%	40.00%	60.00%	80.00%	100.00%

The Courses (criteria) appear again ordered by decreasing significance weight, whereas the students are ranked following the '*Netflows*' (default) ranking rule with student *a*06 first-ranked and student *s*13 last-ranked.

The random outranking relation

```
>>> from outrankingDigraphs import BipolarOutrankingDigraph
>>> g = BipolarOutrankingDigraph(t,Normalized=True)
>>> g.showHTMLRelationTable()
```

Normalized Relation

r(x S y)	s01	s02	s03	s04	s05	s06	s07	s08	s09	s10	s11	s12	s13
s01	–	0.38	0.00	0.62	0.56	-1.00	0.38	-0.50	-0.12	0.50	0.62	0.12	0.81
s02	-0.12	–	0.50	-0.12	-0.44	-0.75	0.00	-0.75	-0.12	0.25	1.00	0.25	0.81
s03	0.12	0.00	–	-0.25	-0.31	-0.75	-0.50	-0.50	-0.25	0.38	1.00	0.12	0.81
s04	0.25	0.50	0.38	–	-0.06	-0.62	0.00	-0.12	-0.38	0.50	1.00	0.38	0.81
s05	0.81	0.69	0.31	0.81	–	-0.56	0.69	-0.06	0.06	0.50	0.81	0.31	0.81
s06	1.00	0.75	1.00	1.00	0.56	–	0.88	0.88	1.00	0.75	1.00	1.00	0.81
s07	0.38	0.62	0.50	0.00	-0.06	-0.75	–	-0.25	0.38	0.38	0.88	0.38	0.69
s08	0.50	0.75	0.50	0.50	0.06	-0.38	0.88	–	0.25	0.75	1.00	0.50	0.81
s09	0.75	0.75	0.25	0.38	0.56	-0.50	0.62	-0.25	–	0.38	0.88	0.75	0.69
s10	-0.50	0.12	-0.38	0.00	-0.50	-0.75	-0.38	-0.75	-0.38	–	0.25	-0.12	0.75
s11	-0.62	-0.25	0.25	-1.00	-0.81	-1.00	-0.88	-1.00	-0.88	-0.25	–	-0.62	0.81
s12	0.38	-0.25	0.38	0.25	-0.06	-0.50	-0.25	-0.50	-0.25	0.12	1.00	–	0.81
s13	-0.81	-0.56	-0.44	-0.81	-0.81	-0.81	-0.69	-0.81	-0.69	-0.75	0.31	-0.81	–

Valuation domain: [-1.00; +1.00]

The best choice recommendation

```
>>> g.showBestChoiceRecommendation()
***********************
Rubis choice recommendation(s)
 (in decreasing order of determinateness)
Credibility domain: [-1.00,1.00]
 === >> potential best student(s)
 * choice              : ['s06']
  independence        : 1.00
  dominance           : 0.38
  absorbency          : -1.00
  covering (%)        : 100.00
  determinateness (%) : 76.92
  - most credible action(s) = { 's06': 0.56, }
    # Condorcet winner
 === >> potential weakest students(s)
 * choice              : ['s11', 's13']
  independence        : 0.31
  dominance           : -0.81
  absorbency          : 0.44
  covered (%)         : 95.45
  determinateness (%) : 81.49
  - most credible action(s) = { 's13': 0.69, 's11': 0.31, }
```