HPC-Ranking big multicriteria performance tableaux

Raymond Bisdorff

Université du Luxembourg
FSTC/ILAS

EURO 2016
Poznan, July 2016

Motivation: showing a performance tableau

Consider a performance table showing the service quality of 12 commercial cloud providers measured by an external auditor on 14 incommensurable performance criteria.

Performance table auditor2_1

criterion	upT	dwT	ouT	LB	MTBF	Rcv	Lat	RspT	Thrpt	stoC	snpC	auT	enC	auD
Amz	2	2	2	4	3	3	NA	3	NA	4	NA	4	4	4
Cen	4	4	0	4	4	4	NA	2	NA	3	NA	4	4	4
Cit	2	4	2	4	3	4	NA	2	NA	3	4	4	4	4
Dig	2	1	4	4	3	3	NA	2	NA	3	NA	4	4	4
Ela	4	4	0	4	4	4	NA	4	NA	3	4	4	4	4
GMO	1	3	4	4	3	2	NA	4	NA	3	NA	4	4	4
Ggl	4	2	1	4	2	3	NA	2	NA	4	4	4	4	4
HP	3	3	2	4	4	3	NA	4	NA	3	4	4	4	4
Lux	2	2	2	4	3	3	NA	2	NA	2	NA	4	4	4
MS	4	4	0	4	4	4	NA	4	NA	4	NA	4	4	4
Rsp	NA	NA	NA	4	NA	3	NA	NA	NA	3	4	4	4	4
Sig	4	4	0	4	4	4	NA	3	NA	3	4	4	4	4

Legend: 0 = *'very weak'*, 1 = *'weak'*, 2 = *'fair'*, 3 = *'good'*, 4 = *'very good'*,*'NA'* = missing data; '*green*' and '*red*' mark the **best**, respectively the **worst**, performances on each criterion.

Motivation: showing an ordered heat map

The same performance tableau may be optimistically colored with the highest 7-tiles class of the marginal performances and presented like a heat map,

Heatmap of Performance Tableau 'auditor2_1'

criteria	stoC	snpC	upT	dwT	ouT	LB	MTBF	Rcv	Lat	RspT	Thrpt	auT	enC	auD
weights	3.00	3.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	1.00	1.00	1.00
MS	4	NA	4	4	0	4	4	4	NA	4	NA	4	4	4
Ela	3	4	4	4	0	4	4	4	NA	4	NA	4	4	4
Sig	3	4	4	4	0	4	4	4	NA	3	NA	4	4	4
Cen	3	NA	4	4	0	4	4	4	NA	2	NA	4	4	4
HP	3	4	3	3	2	4	4	3	NA	4	NA	4	4	4
Cit	3	4	2	4	2	4	3	4	NA	2	NA	4	4	4
Ggl	4	4	4	2	1	4	2	3	NA	2	NA	4	4	4
GMO	3	NA	1	3	4	4	3	2	NA	4	NA	4	4	4
Rsp	3	4	NA	NA	NA	4	NA	3	NA	NA	NA	4	4	4
Amz	4	NA	2	2	2	4	3	3	NA	3	NA	4	4	4
Dig	3	NA	2	1	4	4	3	3	NA	2	NA	4	4	4
Lux	2	NA	2	2	2	4	3	3	NA	2	NA	4	4	4

Color legend:

quantile	0.14%	0.29%	0.43%	0.57%	0.71%	0.86%	1.00%

eventually linearly ordered, following for instance the Copeland ranking rule, from the best to the worst performing alternatives (ties are lexicographically resolved).

Content

How to rank big performance tableaux ?

- The Copeland ranking rule is based on crisp net flows requiring the in- and out-degree of each node in the outranking digraph;
- When the order n of the outranking digraph becomes big (several thousand or millions of alternatives), this requires handling a huge set of n^2 pairwise outranking situations;
- We shall present hereafter a sparse model of the big outranking digraph, where we only keep a linearly ordered list of diagonal quantiles equivalence classes with local outranking content.

Motivation Qantiles sorting Global ranking HPC ranking Conclusion

Performance quantiles

- Let X be the set of n potential decision alternatives evaluated on a single real performance criteria.
- We denote $x, y, ...$ the performances observed of the potential decision actions in X.
- We call quantile $q(p)$ the performance such that $p\%$ of the observed n performances in X are less or equal to $q(p)$.
- The quantile $q(p)$ is estimated by linear interpolation from the cumulative distribution of the performances in X.

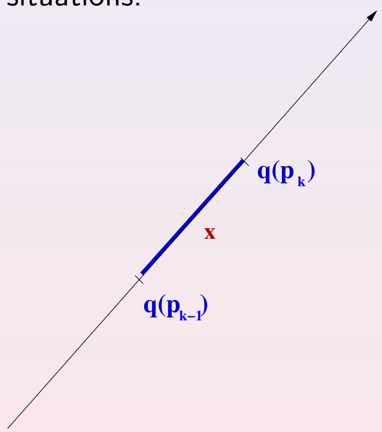
Motivation Qantiles sorting Global ranking HPC ranking Conclusion

Performance quantile classes

- We consider a series: $p_k = k/q$ for $k = 0, ...q$ of $q + 1$ equally spaced quantiles like
  - quartiles: $0, .25, .5, .75, 1$,
  - quintiles: $0, .2, .4, .6, .8, 1$,
  - deciles: $0, .1, .2, ..., .9, 1$, etc
- The upper-closed q^k class corresponds to the interval $]q(p_{k-1}); q(p_k)]$, for $k = 2, ..., q$, where $q(p_q) = \max_X x$ and the first class gathers all data below p_1: $]-\infty; q(p_1)]$.
- The lower-closed q_k class corresponds to the interval $[q(p_{k-1}); q(p_k)[$,for $k = 1, ..., q-1$, where $q(p_0) = \min_X x$ and the last class gathers all data above $q(p_{q-1})$: $[q(p_{q-1}), +\infty[$.
- We call q-tiles a complete series of $k = 1, ..., q$ upper-closed q^k, resp. lower-closed q_k, quantile classes.

q-tiles sorting on a single criteria

If x is a measured performance, we may distinguish three sorting situations:



1. $x \leqslant q(p_{k-1})$ and $x < q(p_k)$
   The performance x is lower than the q^k class;
2. $x > q(p_{k-1})$ and $x \leqslant q(p_k)$
   The performance x belongs to the q^k class;
3. $(x > q(p_{k-1})$ and$)$ $x > q(p_k)$
   The performance x is higher than the p^k class.

If the relation $<$ is the dual of $\geqslant$, it will be sufficient to check that both, $q(p_{k-1}) \not\geqslant x$, as well as $q(p_k) \geqslant x$, are verified for x to be a member of the k-th q-tiles class.

Multiple criteria extension

- $A = \{x, y, z, ...\}$ is a finite set of n objects to be sorted.
- $F = \{1, ..., m\}$ is a finite and coherent family of m performance criteria.
- For each criterion j in F, the objects are evaluated on a real performance scale $[0; M_j]$,

  supporting an indifference threshold ind_j

  and a preference threshold pr_j such that $0 \leqslant ind_j < pr_j \leqslant M_j$.
- The performance of object x on criterion j is denoted x_j.
- Each criterion j in F carries a rational significance w_j such that $0 < w_j < 1.0$ and $\sum_{j \in F} w_j = 1.0$.

Performing marginally *at least as good as*

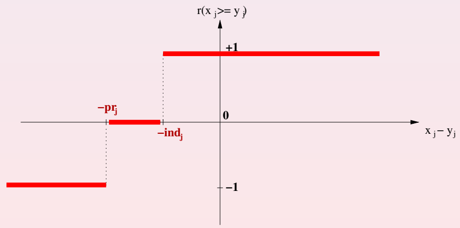
Each criterion j is characterizing a double threshold order $\geqslant_j$ on A in the following way:

$$r(x \geqslant_j y) = \begin{cases} +1 & \text{if} \quad x_j - y_j \geqslant -ind_j \\ -1 & \text{if} \quad x_j - y_j \leqslant -pr_j \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

+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.



Performing globally *at least as good as*

Each criterion j contributes the significance w_j of his "*at least as good as*" characterization $r(\geqslant_j)$ to the global characterization $r(\geqslant)$ in the following way:

$$\boxed{r(x \geqslant y) \;=\; \sum_{j \in F} \left[\, w_j \cdot r(x \geqslant_j y) \,\right]} \tag{2}$$

$r > 0$ signifies x is *globally performing at least as good as* y,

$r < 0$ signifies that x is *not globally performing at least as good as* y,

$r = 0$ signifies that it is *unclear* whether x is globally performing at least as good as y.

The bipolar outranking relation $\succsim$

From an epistemic point of view, we say that:

1. object x *outranks* object y, denoted $(x \succsim y)$, if
   1.1 a significant majority of criteria validates a global outranking situation between x and y, i.e. $(x \geqslant y)$ and
   1.2 no veto $(x \not\lll y)$ is observed on a discordant criterion,
2. object x *does not outrank* object y, denoted $(x \not\succsim y)$, if
   2.1 a significant majority of criteria invalidates a global outranking situation between x and y, i.e. $(x \not\geqslant y)$ and
   2.2 no counter-veto $(x \not\ggg y)$ observed on a concordant criterion.

Polarising the global "*at least as good as*" characteristic

The valued bipolar outranking characteristic $r(\succsim)$ is defined as follows:

$$r(x \succsim y) = \begin{cases} 0, & \text{if } \big[\exists j \in F : r(x \lll_j y)\big] \wedge \big[\exists k \in F : r(x \ggg_k y)\big] \\ \big[\, r(x \geqslant y) \circledvee -r(x \lll y) \,\big] & \text{, otherwise.} \end{cases}$$

And in particular,

- $r(x \succsim y) = r(x \geqslant y)$ if no very large positive or negative performance differences are observed,
- $r(x \succsim y) = 1$ if $r(x \geqslant y) \geqslant 0$ and $r(x \ggg y) = 1$,
- $r(x \succsim y) = -1$ if $r(x \geqslant y) \leqslant 0$ and $r(x \lll y) = 1$,

q-tiles sorting with bipolar outrankings

Proposition

The bipolar characteristic of x belonging to upper-closed q-tiles class q^k, resp. lower-closed class q_k, may hence, in a multiple criteria outranking approach, be assessed as follows:

$$r(x \in q^k) = \min\big[\, -r\big(\mathbf{q}(p_{k-1}) \succsim x\big),\ r\big(\mathbf{q}(p_k) \succsim x\big) \,\big]$$
$$r(x \in q_k) = \min\big[\, r\big(x \succsim \mathbf{q}(p_{k-1})\big),\ -r\big(x \succsim \mathbf{q}(p_k)\big) \,\big]$$

Proof.

The bipolar outranking relation $\succsim$, being weakly complete, verifies actually the coduality principle. The dual ($\not\succsim$) of $\succsim$ is, hence, identical to the strict converse outranking $\precsim$ relation. □

R. Bisdorff (2013), *On polarizing outranking relations with large performance differences*. Journal of Multi-Criteria Decision Analysis **20**:3-12

The multicriteria (upper-closed) q-tiles sorting algorithm

1. **Input**: a set X of n objects with a performance table on a family of m criteria and a set $\mathcal{Q}$ of $k = 1, .., q$ empty q-tiles equivalence classes.
2. **For each** object $x \in X$ **and each** q-tiles class $q^k \in \mathcal{Q}$
   2.1 $r(x \in q^k) \quad \leftarrow \quad \min\big(-r(\mathbf{q}(p_{k-1}) \succsim x), r(\mathbf{q}(p_k) \succsim x)\big)$
   2.2 if $r(x \in q^k) \ \geqslant\ 0$:
   add x to q-tiles class q^k
3. **Output**: $\mathcal{Q}$

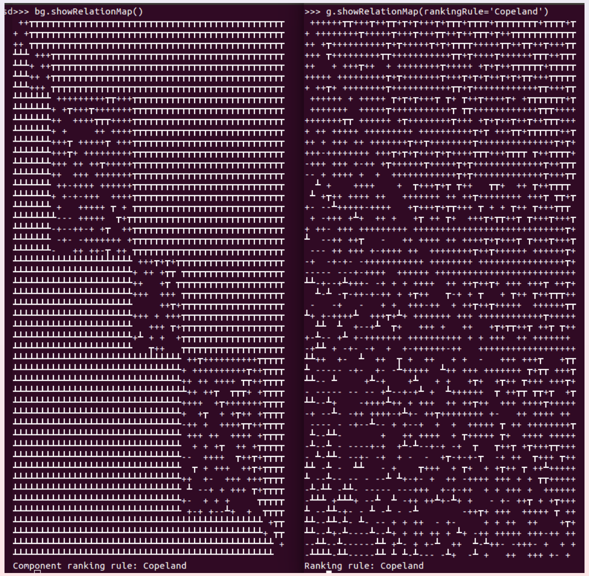
Comment

1. *The complexity of the q-tiles sorting algorithm is $\mathcal{O}(nmq)$; linear in the number of decision actions (n), criteria (m) and quantile classes (q).*
2. *As $\mathcal{Q}$ represents a partition of the criteria measurement scales, i.e. the upper limits of the preceding category correspond to the lower limits of the succeeding ones, there is a potential for run time optimization.*

Example of sparse outranking Digraph

```
>>> from bigOutrankingDigraphs import *
>>> t = RandomPerformanceTableau(numberOfActions=50,seed=5)
>>> bg = BigOutrankingDigraphMP(t,quantiles=5)
>>> bg.showDecomposition()
*--- quantiles decomposition in decreasing order---*
c1. [0.60-0.80[ : ['a22','a24','a32']
c2. [0.40-0.80[ : ['a16', 'a28','a31','a40']
c3. [0.40-0.60[ : ['a01','a02','a05','a06','a10',
                   'a13','a15','a25','a27','a35',
                   'a36','a37','a39','a41','a48']
c4. [0.20-0.60[ : ['a09','a14','a18','a20','a26',
                   'a38','a43','a45','a49']
c5. [0.20-0.40[ : ['a03','a04','a07','a08','a11',
                   'a12','a17','a21','a29','a30',
                   'a33','a34','a42','a44','a47']
c6. [0.00-0.40[ : ['a46','a50']
c7. [0.00-0.20[ : ['a19','a23']
```

Sparse versus standard outranking digraph of order 50



Symbol legend

- ⊤ outranking for certain
- \+ more or less outranking
- ' ' indeterminate
- − more or less outranked
- ⊥ outranked for certain

Sparse digraph *bg*:
Actions : 50
Criteria : 7
Sorted by : 5-Tiling
Ranking rule : Copeland
Components : 7
Minimal order : 1
Maximal order : 15
Average order : 7.1
fill rate : 20.980%
correlation : +0.7563

Content

Properties of *q*-tiles sorting result

1. *Coherence*: Each object is always sorted into a non-empty subset of adjacent *q*-tiles classes.
2. *Uniqueness*: If the *q*-tiles classes represent a discriminated partition of the measurement scales on each criterion and $r \neq 0$, then every object is sorted into exactly one *q*-tiles class.
3. *Independence*: The sorting result for object x, is independent of the other object's sorting results.

Comment

The independence property gives us access to efficient parallel processing of class membership characteristics $r(x \in q^k)$ *for all* $x \in X$ *and* q^k *in* $\mathcal{Q}$.

Ordering the *q*-tiles sorting result

The *q*-tiles sorting result leaves us with a more or less refined partition of the set X of n potential decision actions. For linearly ranking from best to worst the resulting parts of the *q*-tiles partition we may apply three strategies:

1. Optimistic: In decreasing lexicographic order of the upper and lower quantile class limits;
2. Pessimistic: In decreasing lexicographic order of the lower and upper quantile class limits;
3. Average: In decreasing numeric order of the average of the lower and upper quantile limits.

q-tiles ranking algorithm

1. **Input**: the outranking digraph $\mathcal{G}(X, \succsim)$, a partition P_q of k linearly ordered decreasing parts of X obtained by the q-sorting algorithm, and an empty list $\mathcal{R}$.
2. **For each** quantile class $q^k \in P_q$:
   if $\#(q^k) > 1$:
   $R_k \leftarrow$ **locally rank** q^k in $\mathcal{G}_{|q^k}$
   (if ties, render alphabetic order of action keys)
   else: $R_k \leftarrow q^k$
   append R_k to $\mathcal{R}$
3. **Output**: $\mathcal{R}$

q-tiles ranking algorithm – Comments

1. The complexity of the q-tiles ranking algorithm is linear in the number k of components resulting from a q-tiles sorting which contain more than one action.
2. Three local ranking rules are available – *Copeland's*, *Net-flows'* and *Kohler's rule* – of complexity $\mathcal{O}(\#(q^k)^2)$ on each restricted outranking digraph $\mathcal{G}_{|q^k}$.
3. In case of local ties (very similar evaluations for instance), the **local ranking** procedure will render these actions in increasing alphabetic ordering of the action keys.

Content

Multithreading the q-tiles sorting & ranking procedure

1. Following from the independence 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.
2. Furthermore, the **ranking** procedure being local to each diagonal component, these procedures may hence be safely processed in parallel threads on each restricted outranking digraph $\mathcal{G}_{|q^k}$.

Generic algorithm design for parallel processing

```
from multiprocessing import Process, active_children
    class myThread(Process):
        def __init__(self, threadID, ...)
                    Process.__init__(self)
                    self.threadID = threadID
                    ...
        def run(self):
            ... task description
            ...

nbrOfJobs = ...
for job in range(nbrOfJobs):
    ... pre-threading tasks per job
    print('iteration = ',job+1,end=" ")
    splitThread = myThread(job, ...)
    splitThread.start()
while active_children() != []:
    pass
print('Exiting computing threads')
for job in range(nbrOfJobs):
    ... post-threading tasks per job
```


Choosing the right HPC granularity ?

With k single threaded CPUs, is it more efficient:

- to run k simple jobs in parallel ?
- to run in parallel a smaller number of complex jobs ?
- to align the numbers of parallel jobs and tasks to k ?
- to start more parallel threads than available cores ?
- to feed k parallel workers with a shared tasks queue ?
- to split the HPC program in several separate run executables ?

HPC performance measurements

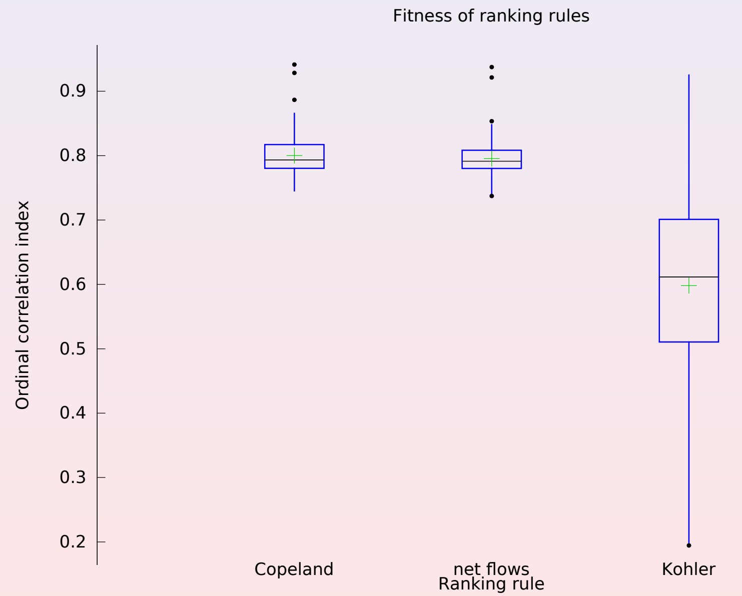
digraph	standard model			sparse model		
order	#c.	t_g sec.	τ_g	#c.	t_{bg}	τ_{bg}
1 000	118	6"	+0.88	8	4"	+0.83
2 000	118	15"	+0.88	8	9"	+0.83
2 500	118	27"	+0.88	8	14"	+0.83
10 000				118	13"	
15 000				118	22"	
25 000				118	39"	
50 000				118	2'	
100 000	(size	=	10^{10})	118	5'	(fill rate = 0.223%)
1 000 000	(size	=	10^{12})	118	1h17'	(fill rate = 0.049%)
1 732 051	(size	=	3×10^{12})	118	3h09'	(fill rate = 0.038%)
2 236 068	(size	=	5×10^{12})	118	4h50'	(fill rate = 0.032%)

Legend:

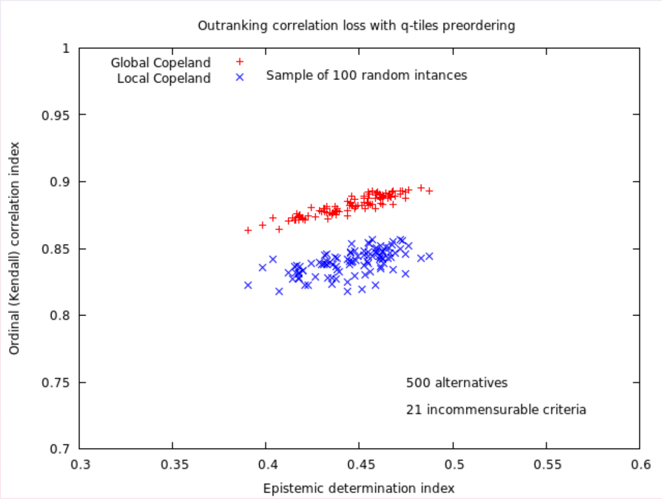
- #c. = number of cores;
- g: standard outranking digraph, bg: the sparse outranking digraph;
- t_g, resp. t_{bg}, are the corresponding constructor run times;
- τ_g, resp. τ_{bg} are the ordinal correlation of the Copeland ordering with the given outranking relation.

Choosing a ranking rule – fitness of ranking rule

Sample of 100 random outranking graphs of order 250

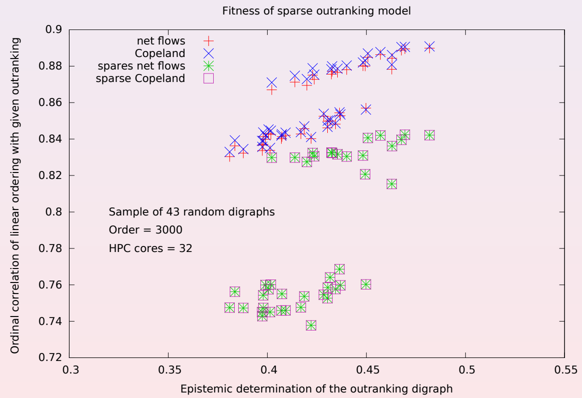


Standard versus 50-tiled sparse outranking digraphs



Profiling the local ranking procedure

It is opportune to use Copeland's rule for ranking from the standard outranking digraph, whereas both, Net Flows and Copeland's ranking rule, are equally efficient on the sparse outranking digraph.



The quality of the sparse model based linear ordering is depending on the alignment of the given outranking digraph, but not on its actual order.

Concluding ...

- We implement a sparse outranking digraph model coupled with a linearly ordering algorithm based on quantiles-sorting & local-ranking procedures;
- Global ranking result fits apparently well with the given outranking relation;
- Independent sorting and local ranking procedures allow effective multiprocessing strategies;
- Efficient scalability allows hence the linear ranking of very large sets of potential decision actions (millions of nodes) graded on multiple incommensurable criteria;
- Good perspectives for further optimization with cPython and HPC ad hoc tuning.

Python and cython HPC modules available under:
http://github.com/rbisdorff/Digraph3

Documentation: http://charles-sanders-peirce.uni.lu/docDigraph3/