# UL HPC Users' session: Mastering big data

Raymond Bisdorff

Université du Luxembourg
FSTC/ILAS

Luxembourg, June 2017



## 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.

| 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,

**Ranking of cloud providers by service quality**

| criteria | dwT | Rcv | MTBF | upT | RspT | stoC | auD | enC | auT | snpC | Thrpt | Lat | LB | ouT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| weights | 2.00 | 2.00 | 2.00 | 2.00 | 2.00 | 3.00 | 1.00 | 1.00 | 1.00 | 3.00 | 2.00 | 2.00 | 2.00 | 2.00 |
| tau(*) | 0.56 | 0.44 | 0.44 | 0.41 | 0.33 | 0.14 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | -0.45 |
| MS | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | NA | NA | NA | 4 | 0 |
| Ela | 4 | 4 | 4 | 4 | 4 | 3 | 4 | 4 | 4 | 4 | NA | NA | 4 | 0 |
| Sig | 4 | 4 | 4 | 4 | 3 | 3 | 4 | 4 | 4 | 4 | NA | NA | 4 | 0 |
| Cen | 4 | 4 | 4 | 4 | 2 | 3 | 4 | 4 | 4 | NA | NA | NA | 4 | 0 |
| HP | 3 | 3 | 4 | 3 | 4 | 3 | 4 | 4 | 4 | 4 | NA | NA | 4 | 2 |
| Cit | 4 | 4 | 3 | 2 | 2 | 3 | 4 | 4 | 4 | 4 | NA | NA | 4 | 2 |
| GMO | 3 | 2 | 3 | 1 | 4 | 3 | 4 | 4 | 4 | NA | NA | NA | 4 | 4 |
| Ggl | 2 | 3 | 2 | 4 | 2 | 4 | 4 | 4 | 4 | 4 | NA | NA | 4 | 1 |
| Rsp | NA | 3 | NA | NA | NA | 3 | 4 | 4 | 4 | 4 | NA | NA | 4 | NA |
| Amz | 2 | 3 | 3 | 2 | 3 | 4 | 4 | 4 | 4 | NA | NA | NA | 4 | 2 |
| Dig | 1 | 3 | 3 | 2 | 2 | 3 | 4 | 4 | 4 | NA | NA | NA | 4 | 4 |
| Lux | 2 | 3 | 3 | 2 | 2 | 2 | 4 | 4 | 4 | NA | NA | NA | 4 | 2 |

*Color legend:*

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

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

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



## 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 use instead a sparse model of the outranking digraph, where we only keep a linearly ordered list of diagonal multicriteria quantiles equivalence classes with local outranking content.

## Example of sparse outranking Digraph

```
>>> from sparseOutrankingDigraphs import *
>>> t = RandomPerformanceTableau(numberOfActions=50)
>>> bg = PreRankedOutrankingDigraph(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**

- $\top$ outranking for certain
- $+$ more or less outranking
- ' ' indeterminate
- $-$ more or less outranked
- $\bot$ 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

## 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. *Separability*: Computing the sorting result for object $x$ is independent from the computing of the other objects' sorting results.

### Comment

*The separability 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 $\mathcal{Q}$.*



## Multithreading the $q$-tiles sorting & ranking procedures

1. 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.
2. Furthermore, the **ranking** procedure being local to each diagonal component, these procedures may as well 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
```



## HPC performance measurements

| digraph order | standard model #c. | $t_g$ sec. | $\tau_g$ | sparse model #c. | $t_{bg}$ | $\tau_{bg}$ |
|---|---|---|---|---|---|---|
| 1 000 | 118 | 6" | +0.88 | 8 | 1.6' | +0.83 |
| 2 000 | 118 | 15" | +0.88 | 8 | 3.5" | +0.83 |
| 2 500 | 118 | 27" | +0.88 | 8 | 4.4" | +0.83 |
| 10 000 | | | | 118 | 7" | |
| 15 000 | | | | 118 | 12" | |
| 25 000 | | | | 118 | 21" | |
| 50 000 | | | | 118 | 48" | |
| 100 000 | (size | = | $10^{10}$) | 118 | 2' | (fill rate = 0.077%) |
| 1 000 000 | (size | = | $10^{12}$) | 118 | 36' | (fill rate = 0.028%) |
| 1 732 051 | (size | = | $3 \times 10^{12}$) | 118 | 2h17' | (fill rate = 0.010%) |
| 2 236 068 | (size | = | $5 \times 10^{12}$) | 118 | 3h15' | (fill rate = 0.010%) |

**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;
- $\tau_g$, resp. $\tau_{bg}$ are the ordinal correlation of the Copeland ordering with the given outranking relation.



## Gaia-80 November 2016 ranking record

```
bisdorff@bisdorff-PC: ~
Results with 118 cores on gaia-80, seed=105
model: Obj, equiobjectives, ('beta', 'variable', None)
Tue Nov 22 07:47:17 2016
perfTab: 625.210357 sec., 5053959984 bytes
*----- show short --------------*
Instance name      : random3ObjectivesPerfTab_mp
# Actions          : 2500000
# Criteria         : 21
Sorting by         : 500-Tiling
Ordering strategy  : average
Local ranking rule : Copeland
# Components       : 200499
Minimal size       : 1
Maximal order      : 543
Average order      : 12.5
Fill rate          : 0.008%
*-- Constructor run times (in sec.) --*
# Threads          : 118
Total time         : 10604.06302
QuantilesSorting   : 6221.00685
Preordering        : 854.70296
Decomposing        : 3528.33810
Ordering           : 0.00007
0 15:37:30 rbisdorff@access(gaia-cluster) Gaia80 $
```



## 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/