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.

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.

The same performance tableau may be optimistically colored with the highest 7-tiles class of the marginal performances and presented like a heat map, eventually linearly ordered, following for instance the Copeland ranking rule, from the best to the worst performing alternatives (ties are lexicographically resolved).

How to handle 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), this requires handling a huge set of $n^2$ pairwise outranking situations;
- We shall present hereafter a sparse model of the outranking digraph, where we only keep a linearly ordered list of quantiles equivalence classes with local outranking content.
Content

1. Sparse model of outranking digraph
   Single criteria quantiles sorting
   Multiple criteria quantiles sorting

2. Ranking a q-tiled performance tableau
   Properties of the q-tiles sorting
   Ordering the q-tiles sorting result
   q-tiles ranking algorithm

3. HPC-ranking a big performance tableau
   Multithreading the sorting&ranking procedure
   Profiling the HPC sorting&ranking procedure

Performance quantiles

- Let \( X \) be the set of \( n \) potential decision alternatives evaluated on a single real performance criteria.
- We denote \( x, y, \ldots \) 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 \( X \).

q-tiles sorting on a single criteria

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

1. \( x \leq q(p_{k-1}) \) and \( x < q(p_k) \)
   The performance \( x \) is lower than the \( q^k \) class;

2. \( x > q(p_{k-1}) \) and \( x \leq q(p_k) \)
   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 \( \geq \), it will be sufficient to check that both, \( q(p_{k-1}) \nless x \), as well as \( q(p_k) \nless x \), are verified for \( x \) to be a member of the \( k\)-th q-tiles class.
Multiple criteria extension

- \( A = \{x, y, z, \ldots\} \) is a finite set of \( n \) objects to be sorted.
- \( F = \{1, \ldots, m\} \) 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 \( \text{ind}_j \) and a preference threshold \( \text{pr}_j \) such that \( 0 \leq \text{ind}_j < \text{pr}_j \leq M_j \).
- 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 \( \geq i \) on \( A \) in the following way:

\[
 r(x \geq y) = \begin{cases} 
 +1 & \text{if } x_j - y_j \geq -\text{ind}_j \\
 -1 & \text{if } x_j - y_j \leq -\text{pr}_j \\
 0 & \text{otherwise.} 
\end{cases}
\]  

(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 \( y \).

The bipolar outranking relation \( \succapprox \)

From an epistemic point of view, we say that:

1. object \( x \) outranks object \( y \), denoted \( (x \succapprox y) \), if
   1.1 a significant majority of criteria validates a global outranking situation between \( x \) and \( y \), i.e. \( (x \geq y) \) and
   1.2 no veto \( (x \not\precapprox y) \) is observed on a discordant criterion,

2. object \( x \) does not outrank object \( y \), denoted \( (x \not\succapprox y) \), if
   2.1 a significant majority of criteria invalidates a global outranking situation between \( x \) and \( y \), i.e. \( (x \not\geq y) \) and
   2.2 no counter-veto \( (x \not\precapprox y) \) observed on a concordant criterion.
Polarising the global “at least as good as” characteristic

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

\[
 r(x \succcurlyeq y) = \begin{cases} 
 0, & \text{if } \exists j \in F : r(x \preccurlyeq j y) \land \exists k \in F : r(x \succcurlyeq k y) \\
 \left[ r(x \geq y) \land \neg r(x \preccurlyeq y) \right], & \text{otherwise.}
\end{cases}
\]

And in particular,
- \( r(x \succcurlyeq y) = r(x \geq y) \) if no very large positive or negative performance differences are observed,
- \( r(x \succcurlyeq y) = 1 \) if \( r(x \geq y) \geq 0 \) and \( r(x \succcurlyeq y) = 1 \),
- \( r(x \succcurlyeq y) = -1 \) if \( r(x \geq y) \leq 0 \) and \( r(x \preccurlyeq y) = 1 \),

\[r(x ≿ y) = 1 \text{ if } r(x ⩾ y) ⩾ 0 \text{ and } r(x ≫ y) = 1,\]
\[r(x ≿ y) = −1 \text{ if } r(x ⩾ y) ⩽ 0 \text{ and } r(x ≪ y) = 1,\]

The bipolar outranking relation \( \succcurlyeq \) is, hence, identical to the strict converse outranking \( ⋨ \) relation.

Example of sparse outranking Digraph

```python
>>> from bigOutrankingDigraphs import *
>>> t = RandomPerformanceTableau(numberOfActions=50, seed=5)
>>> bg = BigOutrankingDigraphMP(t, quantiles=5)
>>> bg.showDecomposition()
```

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 \( Q \) of \( k = 1, \ldots, q \) empty \( q \)-tiles equivalence classes.
2. **For each** object \( x \in X \) and each \( q \)-tiles class \( q^k \in Q \)
   1. \( r(x \in q^k) \leftarrow \min \left( -r(q(p_{k-1}) \succcurlyeq x), r(q(p_k) \succcurlyeq x) \right) \)
   2. if \( r(x \in q^k) \geq 0 \):
      - add \( x \) to \( q \)-tiles class \( q^k \)
3. **Output**: \( Q \)

Comment

1. The complexity of the \( q \)-tiles sorting algorithm is \( O(nmq) \); *linear* in the number of decision actions (\( n \)), criteria (\( m \)) and quantile classes (\( q \)).
2. As \( Q \) 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.
## Sparse versus standard outranking digraph of order 50

### Symbol legend
- $\top$ certainly valid
- $+$ more or less valid
- $'$ indeterminate
- $-$ more or less invalid
- $\perp$ certainly invalid

### 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

1. Sparse model of outranking digraph
   - Single criteria quantiles sorting
   - Multiple criteria quantiles sorting

2. Ranking a $q$-tiled performance tableau
   - Properties of the $q$-tiles sorting
   - Ordering the $q$-tiles sorting result
   - $q$-tiles ranking algorithm

3. HPC-ranking a big performance tableau
   - Multithreading the sorting & ranking procedure
   - Profiling the HPC sorting & ranking procedure

### 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.

### 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 \( G(X, \succeq) \), 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 \( \#(q^k) > 1 \):
     - \( R_k \leftarrow \text{locally rank } q^k \) in \( G|q^k \) (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 \( O((\#(q^k))^2) \) on each restricted outranking digraph \( G|q^k \).

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**

1. Sparse model of outranking digraph
   - Single criteria quantiles sorting
   - Multiple criteria quantiles sorting

2. Ranking a \( q \)-tiled performance tableau
   - Properties of the \( q \)-tiles sorting
   - Ordering the \( q \)-tiles sorting result
   - \( q \)-tiles ranking algorithm

3. HPC-ranking a big performance tableau
   - Multithreading the sorting&ranking procedure
   - Profiling the HPC sorting&ranking procedure

**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 \( G|q^k \).
Generic algorithm design for parallel processing

```python
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 granularity?

Is it more efficient:
- to run many simple jobs in parallel?
- to run in parallel a small number of complex jobs?
- to align the number of parallel jobs to the number of available cores?
- to start more parallel jobs than available cores?

HPC performance measurements

<table>
<thead>
<tr>
<th>digraph order</th>
<th>#c.</th>
<th>t_g sec.</th>
<th>τ_g</th>
<th>#c.</th>
<th>t_bg sec.</th>
<th>τ_bg</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>50</td>
<td>7</td>
<td>+0.88</td>
<td>160</td>
<td>2</td>
<td>+0.83</td>
</tr>
<tr>
<td>1000</td>
<td>50</td>
<td>27</td>
<td>+0.88</td>
<td>160</td>
<td>3</td>
<td>+0.83</td>
</tr>
<tr>
<td>2000</td>
<td>50</td>
<td>108</td>
<td>+0.88</td>
<td>160</td>
<td>7</td>
<td>+0.83</td>
</tr>
<tr>
<td>2500</td>
<td>50</td>
<td>160</td>
<td>+0.88</td>
<td>160</td>
<td>9</td>
<td>+0.83</td>
</tr>
<tr>
<td>10000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 – run time statistics

Sample of 100 random outranking graphs of order 250
Choosing a ranking rule – fitness of ranking rule

Sample of 100 random outranking graphs of order 250

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.

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 ranking of very large sets of potential decision actions (thousands of nodes) graded on multiple incommensurable criteria;
- Good perspectives for further optimization and ad-hoc fine-tuning.