Motivation: showing a performance tableau

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

Legend: $0 = \text{very weak}$, $1 = \text{weak}$, $2 = \text{fair}$, $3 = \text{good}$, $4 = \text{very good}$, '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, eventually linearly ordered, following for instance the Copeland ranking rule, from the best to the worst performing alternatives (ties are lexicographically resolved).

Content

1. Sparse model of big outranking digraph
   How to rank big performance tableaux?
   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
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.

Performance quantile classes

- We consider a series: $p_k = k/q$ for $k = 0, \ldots, q$ 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 $k = 2, \ldots, q$, where $q(p_0) = \max_{X} x$ 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 $k = 1, \ldots, q - 1$, 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 $k = 1, \ldots, q$ 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. $x \leq q(p_{k-1})$ 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, $q(p_{k-1}) \not\geq x$, as well as $q(p_k) \geq x$, 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 $ind_j$ and a preference threshold $pr_j$ such that $0 \leq ind_j < pr_j \leq M_j$.
- 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 globally at least as good as

Each criterion $j$ contributes the significance $w_j$ of his “at least as good as” characterization $r(\geq_j)$ to the global characterization $r(\geq)$ in the following way:

$$ r(x \geq y) = \sum_{j \in F} [w_j \cdot r(x \geq_j y)] $$

(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. $(x \geq y)$ and
   1.2 no veto $(x \not\succ y)$ is observed on a discordant criterion,
2. object $x$ does not outrank object $y$, denoted $(x \not\succ y)$, 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 $(x \not\succ y)$ is observed on a concordant criterion.
Polarising the global “at least as good as” characteristic

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

$$r(x \succeq y) = \begin{cases} 0, & \text{if } \exists j \in F : r(x \ll j y) \land \exists k \in F : r(x \gg k y) \\ [r(x \geq y) \land -r(x \ll y)] & \text{otherwise.} \end{cases}$$

And in particular,

- $r(x \succeq y) = r(x \geq y)$ if no very large positive or negative performance differences are observed,
- $r(x \succeq y) = 1$ if $r(x \geq y) \geq 0$ and $r(x \gg y) = 1$,
- $r(x \succeq y) = -1$ if $r(x \geq y) \leq 0$ and $r(x \ll y) = 1$,
- $r(x \succeq y) = -1$ if $r(x \geq y) \leq 0$ and $r(x \ll y) = 1$.

Example of sparse outranking Digraph

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

The multicroiteria (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 $k=1,..,q$ empty $q$-tiles equivalence classes.

2. **For each** object $x \in X$ and each $q$-tiles class $q^k \in Q$
   1. $r(x \in q^k) \leftarrow \min \{ -r(q(p_k-1) \succeq x), r(q(p_k) \succeq x) \}$
   2. If $r(x \in q^k) \geq 0$:
      1. **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.
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 $Q$. 
### Motivation

#### Qantiles sorting

Global ranking

HPC ranking

Conclusion

#### q-tiles ranking algorithm

1. **Input**: the outranking digraph $G(X, \succsim)$, 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 $G_{|q^k}$ (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 big outranking digraph
   - How to rank big performance tableaux?
   - 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 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

<table>
<thead>
<tr>
<th>digraph order</th>
<th>standard model</th>
<th>sparse model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#c.</td>
<td>$t_g$ sec.</td>
</tr>
<tr>
<td>1000</td>
<td>118</td>
<td>6&quot;</td>
</tr>
<tr>
<td>2000</td>
<td>118</td>
<td>15&quot;</td>
</tr>
<tr>
<td>2500</td>
<td>118</td>
<td>27&quot;</td>
</tr>
<tr>
<td>10000</td>
<td>118</td>
<td>13&quot;</td>
</tr>
<tr>
<td>15000</td>
<td>118</td>
<td>22&quot;</td>
</tr>
<tr>
<td>25000</td>
<td>118</td>
<td>2&quot;</td>
</tr>
<tr>
<td>50000</td>
<td>118</td>
<td>2&quot;</td>
</tr>
<tr>
<td>100000</td>
<td>(size = $10^{10}$)</td>
<td>118</td>
</tr>
<tr>
<td>1000000</td>
<td>(size = $10^{12}$)</td>
<td>118</td>
</tr>
<tr>
<td>1732051</td>
<td>(size = $3 \times 10^{12}$)</td>
<td>118</td>
</tr>
<tr>
<td>2236068</td>
<td>(size = $5 \times 10^{12}$)</td>
<td>118</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;
- $\tau_g$, resp. $\tau_{bg}$, are the ordinal correlation of the Copeland ordering with the given outranking relation.
Motivation | Qantiles sorting | Global ranking | HPC ranking | Conclusion
---|---|---|---|---

### 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/rbisdorf/Digraph3](http://github.com/rbisdorf/Digraph3)