Ten+ Years of Benchmarking with COCO/BBOB

Nikolaus Hansen
Inria
CMAP, CNRS, Ecole Polytechnique, Institut Polytechnique de Paris, France

Presented at the Lorentz Center Workshop Benchmarked: Optimization meets Machine Learning, Leiden 2020
COCO — Comparing Continuous Optimisers

• is a (software) platform for comparing continuous optimisers in a black-box scenario
  https://github.com/numbbo/coco

• *automatises* the tedious and repetitive task of *benchmarking numerical optimisation algorithms in a black-box setting*

• advantage: saves time and *prevents* common (and not so common) *pitfalls*

COCO provides

• experimental and measurement *methodology*  
  main decision: what is the end point of measurement

• suites of benchmark functions  
  single objective, bi-objective, noisy, mixed-integer, more to come…

• *data* of already benchmarked algorithms to compare with
Benchmarking: Related Goals

1. Understanding algorithms
2. Measuring performance in a systematic way (a performance “profile”)
3. Running a competition
Benchmarking: The Global Picture

Two *surprisingly (but not completely) independent* puzzles to solve

- **What to benchmark**: for example, which collection of test problems?
  - experimental setup
  - data collection
  - measures used and presented

- **How to assess performance**?
COCO/BBOB: The Global Picture

User-provided solver
- C/C++ interface
- Python interface
- Java interface
- Matlab/Octave interface

Test suites:
- bbob
- bbob-biobj

Logging functionality

Results of the user-provided solver

Results of other solvers

COCO post-processing

Logging functionality

Table

Charts

...feel free to ask questions...
COCO/BBOB: Test Suite(s)

- Functions are
  - Based on known (analytical) functions, modelling a “known” difficulty
  - Comprehensible
  - Scalable
  - Difficult (also: non-separable)

- Quasi-randomized as instances
  - compared to the “typical standard” (at that time)
  - with arbitrary shifts and smallish irregularities to avoid artificial exploits and mitigate overfitting, emulates repetition of experiments

- The bad
  - Rastrigin function type is somewhat overrepresented
  - 10% of the default targets for F23 Katsuuras are trivial to hit

- Require to define target values (function + target = problem)
  - partly due to function pairing
  - evaluating the domain middle at first is a good “algorithm”
  - natural targets in the discrete search domain are known fitness levels and the global optimum, we may need experiments to define useful targets

Nikolaus Hansen, Inria, IP Paris
with hindsight 20/20

• The good:
  • scattered experiments can be “merged” (and “unmerged”) with a single “drag-and-drop”
  • separation between .info (meta- and summary-data) and .dat files is helpful
  • 10+ years old data are still smoothly usable
  • backwards compatible adjustments are/were possible

• The bad:
  • slightly too few targets (too coarse discretization, not a format issue though with backward compatible fix)
  • “handling” of restarts is suboptimal
  • meta-data are not json-style (key-value Python-dict-style) formatted
  • COCO maintains/writes two somewhat incompatible formats
...feel free to ask questions...
COCO/BBOB: Performance Assessment

“quality indicator” versus “time”
convergence graphs

is all we have (and all we use)
Specifically

- **time**: we use number of function evaluations is invariant under changes of computer hardware, OS, programming language, compiler, …

- **quality indicator**: 

  - SO: affine transformation of the function value (to be minimized) different for each instance

  - MO: negative hypervolume value after objective-wise affine transformation (to be minimized)

Affine transformations are considered as part of the function definition (benchmark suite definition)

they also affect the target values that define a problem: target precisions are defined identical for all functions in a suite
Convergence Graphs is All We Have

- a convergence graph
- lower envelope (a monotonous graph)

we only use the lower envelope
Discretization: Two Possibilities

- a convergence graph
- lower envelope (a monotonous graph)

- **vertical**: by evaluation is a natural discretization
  for wall clock or CPU time we would need to determine discretization intervals

- evaluations are the independent variable
  function value is the dependent variable, the measurement
Discretization: Two Possibilities

- a convergence graph
- lower envelope (a monotonous graph)

- **horizontal**: not a “natural” discretization

- function “target” values are the independent variable
  
  time is the dependent variable, the measurement

- still recovers the original data
  
  a time measurement for each discretization function value, these measurements can be plotted as ECDF

we need to determine discretization intervals
Runtime distribution from a single graph
Runtime distribution from a single graph

the ECDF recovers the monotonous graph

AKA runtime distribution
the ECDF recovers the monotonous graph, discretised and flipped

AKA runtime distribution
• recovering the convergence graph from discretized data
• collecting runtimes from a single experiments as ECDF are two interpretations of the same thing

the ECDF recovers the monotonous graph, discretised and flipped
the ECDF recovers the monotonous graph, discretised and flipped.

the area over the ECDF curve is the average runtime (the geometric average if the x-axis is in log scale).
COCO/BBOB

uses only

horizontal discretization
this is not just a technical subtlety because it crucially determines what measurement we are looking at in the end
COCO/BBOB: Fixed Target(s) versus Fixed Budget

- Leads to different imprecise data in both cases
  - “too” bad performance
    then the data only provide a lower bound estimate for the runtime (and a fixed budget measure at maximum budget)
  - “too” good performance
    (reached global optimum up to the relevant or numerical precision before the given budget)
COCO/BBOB: Fixed Target(s) versus Fixed Budget

The resulting measurement

- Fixed budget (vertical) design: function values
- Fixed target design: evaluations
The fixed budget (vertical) design is (much) easier to set up:

choosing a budget is simpler than choosing a target and we need to chose a maximal “timeout” budget either way.

For the (very) same reason, results from the fixed target (horizontal) design results are (much) simpler to interpret and more conclusive:

without specific insight, a function value is impossible to interpret beyond ordering.

Fixed target results are “budget-free”:

we can compare results run with different maximal “timeout” budgets.

Fixed target results can be meaningfully aggregated in ECDFs and geometric averages:

whereas function values from different functions are not commensurable.
Scales of Measurement ("Quality" of Data)

- Nominal - categorial, define a classification
- Ordinal - define an order, ranks, function values (fixed budget)
- Interval - differences are meaningful
- Rational - ratios are meaningful, we can take the logarithm, time (function evaluations, fixed target)

CAVEAT: mathematical and semantic treatment of data is not the same. From a classification with values \{1, 2\} we can \textit{mathematically} take differences and ratios of the values, but they have no meaningful \textit{semantic interpretation}. 

Treating Success Probabilities
Solving the fast-versus-successful comparison dilemma

\[ p_s(\text{Algo A}) \ll 1, \text{ fast convergence} \]

\[ p_s(\text{Algo B}) \approx 1, \text{ slow convergence} \]
Treating Success Probabilities
Solving the fast-versus-successful comparison dilemma

Short answer: consider as runtime

\[
\text{something} \propto \frac{1}{p_{\text{success}}}
\]

that is, roughly,

\[
\text{runtime} \propto \frac{1}{p_{\text{success}}}
\]
We can **simulate a runtime distribution** by simulated (artificial) restarts using the given independent runs

**Treating Success Probabilities**

Solving the fast-versus-successful comparison dilemma

We can simulate a runtime distribution by simulated (artificial) restarts using the given independent runs.

**Algo Restart A:**

\[ p_s(\text{Algo Restart A}) = 1 \]

**Algo Restart B:**

\[ p_s(\text{Algo Restart B}) = 1 \]

Caveat: the performance of algorithm A critically depends on termination methods (before to hit the target)

*which reflects the situation on a practical problem* unless many runs can be done in parallel
Treating Success Probabilities
Solving the fast-versus-successful comparison dilemma

Replacing the success probability with the expected runtime (ERT, aka Enes, SP2, aRT) to hit a target value in #evaluations is computed (estimated) as:

\[
ERT = \frac{\#\text{evaluations(unti}l\text{ hit the target)}}{\#\text{successes}}
\]

odds ratio

\[
= \text{avg}(\text{evals}_{\text{succ}}) + \frac{N_{\text{unsucc}}}{N_{\text{succ}}} \times \text{avg}(\text{evals}_{\text{unsucc}})
\]

\[
\approx \text{avg}(\text{evals}_{\text{succ}}) + \frac{N_{\text{unsucc}}}{N_{\text{succ}}} \times \text{avg}(\text{evals}_{\text{succ}})
\]

\[
= \frac{N_{\text{succ}} + N_{\text{unsucc}}}{N_{\text{succ}}} \times \text{avg}(\text{evals}_{\text{succ}})
\]

\[
= \frac{1}{\text{success rate}} \times \text{avg}(\text{evals}_{\text{succ}})
\]

defined (only) for #successes > 0. The last three lines are AKA Q-measure or SP1 (success performance).

See [Price 1997] and [Auger&Hansen 2005]
**Data Sets and Usage Statistics**

<table>
<thead>
<tr>
<th>Data Sets Online</th>
<th>BBBOB Suite</th>
<th>227</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>bbo - noisy suite</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>bbo - biobj suite</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>bbo - largescale suite</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>bbo - mixint suite</td>
<td>4</td>
</tr>
<tr>
<td>BBOB Workshop Papers using COCO</td>
<td>143</td>
<td></td>
</tr>
<tr>
<td>Unique Authors on the Workshop Papers</td>
<td>109 from 28 countries</td>
<td></td>
</tr>
<tr>
<td>Papers in Google Scholar Found with the Search Phrase</td>
<td>559</td>
<td></td>
</tr>
<tr>
<td>&quot;comparing continuous optimizers&quot; OR &quot;black-box optimization benchmarking (BBOB)&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Citations to the COCO Documentation Including</td>
<td>1,455</td>
<td></td>
</tr>
</tbody>
</table>

Any `cocopp.archiving.create(folder)`-ed data sets provided under an URL can be loaded with `av = cocopp.archiving.get(URL)` and used in the data processing. See [Hansen et al 2020].