BBOB Black-Box Optimization Benchmarking with COCO (COmparing Continuous Optimizers)
Black-Box Optimization (Search)

Minimize (or maximize) a continuous domain objective (cost, loss, error, fitness) function

\[ f : \mathbb{R}^d \rightarrow \mathbb{R} \]

in a black-box scenario (direct search)

\[ x \rightarrow \bigcirc \rightarrow f(x) \]

where

- gradients are not available or useful
- problem specific knowledge is used only within the black box, e.g. with an appropriate encoding

The search costs are the number of function evaluations
Two objectives:

- Find solution with a smallest possible function value

- With the least possible search costs (number of function evaluations)

- Pareto front is the convergence graph of the optimal algorithm
Why do we need benchmarking?

• putting algorithms to a *standardized* test
  – simplify judgement
  – simplify comparison
  – regression test/quality check under algorithm changes

• algorithm selection

• understanding of algorithms
BBOB in practice (for dummies)
This is the COCO download page.

Last release: 30/05/2012 v11.06

BBOB (5MB) is all that is needed to run the benchmarking experiments and compile a template paper (gathering post-processed results).

BBOB (35MB) contains all files, as listed below.

- **CODE:**
  - tar code in Matlab/Octave to run experiments
  - tar code in C to run experiments
  - tar code in Java to run experiments
  - tar code in Python to run experiments and post-processing and latex templates (3MB)
  - tar R package to run experiments

- **DOCS:**
  - pdf description of experimental procedure
  - pdf (12MB) noiseless functions documentation with figures
  - pdf noiseless functions documentation, version without figures
  - pdf (19MB) noisy function documentation with figures
  - pdf noisy function documentation, version without figures
  - pdf software user documentation
  - html online post-processing package documentation

BUGS for older versions:

- Bugs in version 11.05:
BBOB in practice
BBOB in practice
Matlab script:

```matlab
for dim = [2,3,5,10,20,40]  % small dimensions first, for CPU reasons
    for ifun = benchmarks('FunctionIndices')  % or benchmarksnoisy(...)
        for iinstance = [1:5, 1:5, 1:5]  % first 5 fct instances, three times
            fgeneric('initialize', ifun, iinstance, datapath);

            MY_OPTIMIZER('fgeneric', dim, ...  % necessary parameters
                fgeneric('ftarget'));  % optional termination parameter

            fgeneric('finalize');
        end
    disp(['    date and time: ' num2str(clock, ' %Of')]);
end
    disp(sprintf('---- dimension %d-D done ----', dim));
end
```
Post-processing at the OS shell:

```python
codepath/bbob_pproc/run.py
datapath/latex
templateACMArticle.tex
dvipdf templateACMArticle.dvi
```
Submitted Data Sets

- 2009: 31 noiseless and 21 noisy “data sets”
- 2010: 24 noiseless and 16 noisy “data sets”
- 2012: 30 noiseless and 4 noisy “data sets”
- **Algorithms**: RCGAs (e.g. plain, PCX), EDAs (e.g. IDEA), BFGS & (many) other “classical” methods, ESs (e.g. CMA), PSO, DE, Ant-Stigmergy Alg, Bee Colony, EGS, SPSA, Meta-Strategies...
Components of CoCO

- BBO function testbeds (currently two)
  determine the “scientific question”

- experimental protocol
  important in the details, future changes are unlikely

- data writing/storage protocol
  to be adapted/extended for noisy, constraint & MO case
  long-term data format needs to be determined

- data post-processing and presentation
  continually evolving and improving
  to be adapted/extended for noisy, constraint & MO case
BBOB: the noiseless functions

functions are not perfectly symmetric and are locally deformed

24 functions within five sub-groups

- **Separable** functions
- **Essential unimodal** functions
- **Ill-conditioned** unimodal functions
- **Multimodal structured** functions
- **Multimodal** functions with weak or without structure
BBOB: the noisy functions

three noise-”models”, so-called:

- Gauss, Uniform (severe), Cauchy (outliers)
- Utility-free noise

\[ E(f(x)) \leq E(f(y)) \Rightarrow U(f(x)) \leq U(f(y)) \quad \forall x, y, U \]

30 functions with three sub-groups

- 2x3 functions with weak noise
- 5x3 unimodal functions
- 3x3 multimodal functions
How should we measure performance?
Evaluation of Search Algorithms

needs

- Meaningful **quantitative measure** on benchmark functions or real world problems

- Account for **meta-parameter tuning**
  
  tuning to specific problems can be quite expensive

- Account for **invariance properties**

  prediction of performance is based on "similarity", ideally equivalence classes of functions

- Account for **algorithm internal costs**

  often negligible, depending on the objective function cost
convergence graphs is all we have to start with
Two objectives:

- Find solution with a smallest possible function value
- With the least possible search costs (number of function evaluations)
- For measuring performance: fix one and measure the other
Measuring Performance from Convergence Graphs

fixed-cost versus fixed-target

(best achieved) function value

number of function evaluations (time)
Evaluation of Search Algorithms
Behind the scene

A performance should be

- **quantitative** on the ratio scale (highest possible)
  
  + “algorithm A is two times better than algorithm B” is a meaningful statement
  
  + can assume a wide range of values

- **meaningful** (interpretable) with regard to the real world
  
  possible to transfer from benchmarking to real world

Runtime is the prime candidate (we don't have many choices anyway)
Fixed-target: Measuring Runtime

We measure runtime in number of function evaluations

- as a distribution of runtimes
- as expected runtime ERT

For success probability $0 < p < 1$: (simulated) restarts until a successful run is observed.

$$RT = RT_{\text{succ}} + \sum RT_{\text{unsucc}}$$

$$\approx E(RT_{\text{succ}}) + \frac{1 - p}{p} E(RT_{\text{unsucc}})$$

Feature&drawback: termination method for unsuccessful trials can be critical
away from the fixed-cost scenario, because only the fixed-target scenario gives results that are

- **quantitative** (ratio scale) and
- reasonably well **interpretable**
- missing data ↔ bad algorithms

Disadvantages

- experimental setup is more “complex”
  
  burden is shifted from interpretation to setup

- data collection/presentation is “more intricate”
The data we use

- Currently: samples
  points that the algorithm evaluates

- In future: recommendations or samples
  points the algorithm proposes as solution to the search problem (in each time step)

results in a sequence (or two) of fitness function values
The performance measure we use

- First hitting time to a given target function value in number of fitness function evaluations
  
  equivalent to first hitting time of a sublevel set in search space
We make two implicit assumptions

- algorithms are any-time
  one run can serve to evaluate performance for each time step and/or each target value
- the “true performance” of an algorithm improves with further evaluations
  is monotonous in the number of function evaluations
A Convergence Graph

- as a single graph violates the second assumption for >1000 fevals
First hitting time is monotonous

- first hitting time: a monotonous graph
• another convergence graph
another convergence graph with hitting time
- a target value delivers two data points
a target value delivers two data points
ECDF with four data points
- reconstructing a single run
50 equally spaced targets
the ECDF recovers the monotonous graph
the ECDF recovers the monotonous graph, discretised and flipped
the ECDF recovers the monotonous graph, discretized and flipped
the ECDF recovers the monotonous graph, discretized and flipped.

the area over the ECDF curve is the average log runtime (or geometric average runtime)
15 runs
15 runs

function value vs. $\log_{10}(\text{function evaluations})$
the ECDF of run lengths (runtimes)
function value

$\log_{10}(\text{function evaluations})$

15 runs
15 runs
50 targets
15 runs
50 targets
ECDF
15 runs integrated in a single graph
empirical cumulative distribution functions

- recover a single convergence graph
- **can aggregate** over any set of functions and target values

  they display a set of run lengths or runtimes (RT)

- for a single problem (function & target value) allow to estimate **any statistics** of interest from them, like median, expectation (ERT), … in a meaningful way
 Critics

- the samples (evaluated solutions) do not reflect the real return value of the algorithm
  conjecture: sampling can be consistently done far away from the estimated optimum

- in noisy environments the first hitting time is unrealistic, because the graph is not monotonous
  conjecture: lucky punch is no rare exception
Questions?