Online Black-box Algorithm Portfolios for Continuous Optimization

Petr Baudiš and Petr Pošík baudipet@fel.cvut.cz

Department of Cybernetics, Czech Technical University, Prague

Goal: Find minimum of a black-box function that we haven't seen before, when we have multiple optimization algorithms available.

Problem: How to switch between the available algorithms so that we don't reach the minimum much slower than the best algorithm?

Contribution: Algorithm selection strategy with only small slowdown and better performance stability compared to any fixed algorithm.

Background

Algorithm Portfolios

Often, we have multiple heuristic algorithms, each suited to a different class of problems. *Algorithm Portfolios* aim to combine them within a single general solver that will choose the best suited algorithm for each input.

For each problem instance on input, we apply a *selection strategy* to pick an algorithm from this portfolio:

- Once or along a fixed schedule *(offline selection)* based on one-time measured features.
- In multiple rounds *(online selection)* allocating time based on their previous performance.

These approaches have not yet been really combined. Here, we focus on the online selection strategies.

Black-box Optimization

Continuous black-box optimization solves the problem of finding a minimum value of a continuous function without accessible analytical form.

Vast applications range from operations research to machine learning. Many algorithms are available — simplex algorithms, gradient descent methods or population-based methods.

The de-facto standard for benchmarking optimization methods is the *COmparing Continuous Optimisers* COCO platform. It provides the infrastructure, glue code for both running experiments and preparing high quality figures, a set of common reference results and the code for a set of benchmark functions.

Applying algorithm portfolios on continuous black-box optimization is still a fresh area of research. The main results so far lie either in population methods, combining a variety of genetic algorithms together in non-black-box fashion, or in offline methods based chiefly on exploratory landscape analysis.

Previously, we developed **COCOpf**: an open source Python framework for easy development and benchmarking of selection strategies for algorithm portfolios.

Our Reference Portfolio

We have chosen the six stock minimizers provided by SciPy v0.13:

- **Nelder-Mead**, the Simplex algorithm.
- **Powell**, the tweaked Powell's conjugate direction method.
- **CG**, the nonlinear conjugate gradient Fletcher-Reeves method.
- **BFGS**, the quasi-Newton method of Broyden, Fletcher, Goldfarb and Shanno.
- **L-BFGS-B**, the limited-memory variant of BFGS with box constraints.
- **SLSQP**, the Sequential Least SQuares Programming with box constraints.

These are local minimizers, therefore we use a SciPy wrapper of the **Basin Hopping** restart strategy; conceptually similar to Simulated Annealing with a fixed temperature.

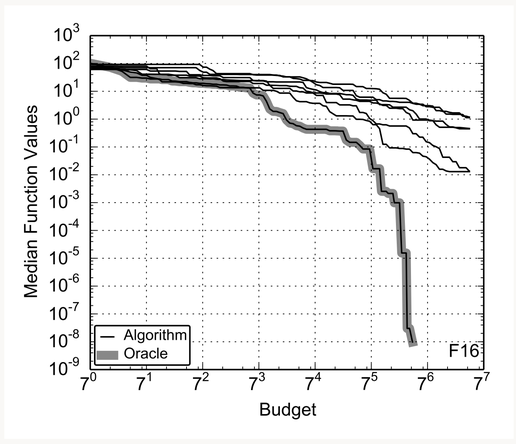
We also included the popular **CMA** algorithm (genetic algorithm with the Covariance Matrix Adaptation evolution strategy). It converges slowly on reasonable functions but it can beat even many difficult targets.

We use portfolio size $|\mathrm{PF}| = 7$ as the exponent base in slowdown measurements.

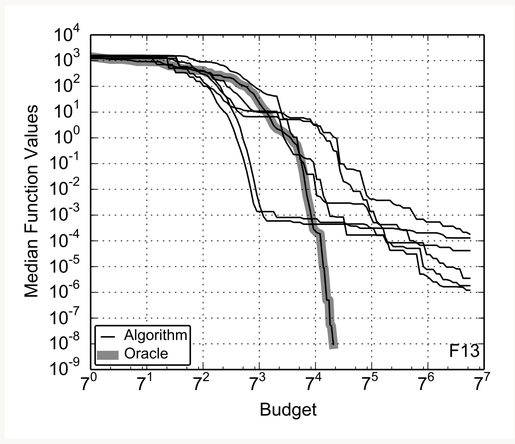
Portfolio Behavior

Based *just on performance* up to a point, does a strategy have enough information on which algorithm is worth further investment? *Sometimes!*

Function classification by convergence:



Stable function (easy)



Volatile function (hard)

Selection Strategies

We implemented various selection strategies — Probability Matching and Adaptive Pursuit, Threshold Ascent, MetaMax variants and UCB1 with Sum-of-Ranks and Area-Under-the-Curve rewards. They all performed worse than those listed below.

In every round, a strategy selects an algorithm and iterates it once:

- **RR** samples algorithms in *round robin*.
- **EG** follows the *epsilon-greedy policy*. The algorithm with the currently best solution is run with $p = 0.5$, a randomly chosen algorithm otherwise.
- **LUCB** follows the *UCB1* MAB policy, where the reward is the EWMA log-rescaled current best function value.
- **RUCB** follows the *UCB1* MAB policy, where the reward is the EWMA rank of the algorithm (when sorted by their current best function value).

The UCB1 Policy

Algorithm portfolio strategy is related to the *multi-armed bandit problem* (MAB). We iterate choices that bring stochastic reward based on a previously unknown but "roughly stationary" distribution.

$$\pi_{\mathrm{UCB1}}(n) = \mathrm{argmax}_i \left(\widehat{\mu}_i(n) + c\sqrt{\frac{2\ln n}{T_i(n)}} \right)$$

UCB1 minimizes cumulative regret. Therefore, here we minimize the (rescaled) sum of all sampled function values.

Typically, $c = 2$ or much less is used. We use $c = 16$ (high exploration bias) because of *volatile* functions.

LUCB function value log-rescaling: When converting arbitrarily raned function value to $\widehat{\mu}_i \in [0, 1]$, we assume that we are just short of optimum, and that in general we converge exponentially fast.

$$f_{opt?} = \min f - \Delta f \qquad \Delta f = 10^{-8}$$

$$g_i = \log(f_i - f_{opt?}) \qquad \mu_i = 1 - \frac{g_i - \min g}{\max g - \min g}$$

Portfolio Performance

Average $\log_{|\mathrm{PF}|}$ slowdown of some algorithms and the selection strategies on function classes:

Solver	all	multi	single	volatile	stable	CMA-good	CMA-bad
CMA	0.7	1.1	0.6	0.5	1.1	0.0	1.5
BFGS	1.5	0.7	1.8	1.4	1.7	2.2	0.8
LUCB	0.9	0.9	0.9	1.3	**0.3**	1.1	0.8
RUCB	1.3	0.9	1.4	1.4	1.1	1.7	0.8
EG	1.4	1.0	1.6	1.6	1.1	2.0	0.9
RR	1.5	1.1	1.7	1.8	1.2	2.1	1.0

Conclusions:

- On *stable* functions, a strategy is much better (on average over multiple functions) than using *any* single fixed algorithm!
- In general, portfolio strategy brings in better runtime balance. CMA is slightly better than a portfolio on average, but is much slower on functions where it doesn't converge.

Future Work

The currently chosen reference portfolio is somewhat ad hoc and unbalanced with CMA dominating in many functions. This makes it actually an interesting testbed, but we need to add more high performance algorithms anyway.

Dream team: BIPOP-CMA, NEWUOA, LineSearch-fminbnd, LineSearch-STEP.

Performance modelling in the style of the *MultiEA* or *GambleTA* algorithms could improve performance-based predictions.

Lesson from *volatile functions*: purely uninformed portfolios are clearly limited. UCB1 can be naturally amended with prior information — for example exploratory landscape analysis using machine learning.

Acknowledgements

This research is supported by the CTU grant SGS14/194/OHK3/3T/13 "Automatic adaptation of search algorithms".