Algorithm Tuning Using Optimization

Philippe Toint (with Margherita Porcelli)

Namur Center for Complex Systems (naXys), University of Namur, Belgium

(philippe.toint@fundp.ac.be)

ISMP, Pittsburgh, July 2015

The context

Two common preoccupations in algorithm design/usage:

- For algorithms designers:

  > How to tune the parameters of an algorithm in order to ensure the best possible performance on the *largest possible class* of applications?

- For algorithm/code users:

  > How to tune the parameters of a code in order to ensure the best possible performance on a *specialized class* of applications?

Does achieving the first does help the second?

A way out ?

Some flexibility is needed !

- Provide a tuning methodology which is applicable to many algorithms
- Provide code which allows user-tuning for his/her pet problem class

$\Longrightarrow$ optimization?

- Need to define an objective function
  (how to measure algorithm performance in this context?)
- Need to define the constraints (on algorithmic parameters)
  - simple bounds (algorithm dependent)
  - continuous/integer/categorical variables + mix
    (ex: blocking size, model type, ...)

Which objective function?

Assume that the (negative) performance perf(params, prob) can be measured by running the considered algorithm with parameters params on problem prob.

- First model: optimize the total/average performance (AO, OPAL):

$$\min_{\text{params}} \sum_{\text{problems}} \text{perf}(\text{params}, \text{prob})$$

- Second model: optimize the robust performance (RO):

$$\min_{\text{params}} \max_{\text{perturbed params}} \sum_{\text{problems}} \text{perf}(\text{perturbed params}, \text{prob})$$

where

$$0.95 * \text{params} \leq \text{perturbed params} \leq 1.05 * \text{params}$$

A new tool: BFO (the Brute Force Optimizer)

BFO: a new *local* optimization package with

- randomized pattern search methodology
  (does not require continuity of the objective function)
- allows bounds on the variables
- allows continuous/discrete or mixed integer variables
- handles multilevel/equilibrium problems
  (needed for the robust tuning strategy)
- includes self-tuning facilities

BFO self tuning

BFO has been self-tuned

- on a large set of test problems (CUTEst) with continuous and mixed-integer variables
- using both the average and robust tuning strategies
- for all 7 algorithmic parameters

Outcome :

- robust strategy slightly better
- gains in performance of
  - 30% for continuous problems
  - 19% for mixed-integer problems

  compared with "intuitively reasonable values"
- *very* competitive with NOMAD (state-of-the-art pattern search algo)

And then...

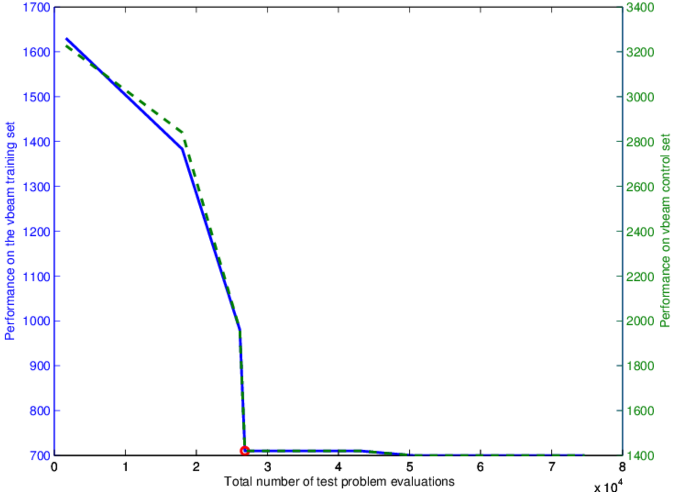
... the algorithm designer is (hopefully) happy !
But what about the user (with his/her own specific problems)?

BFO allows training by the user for specific problem classes

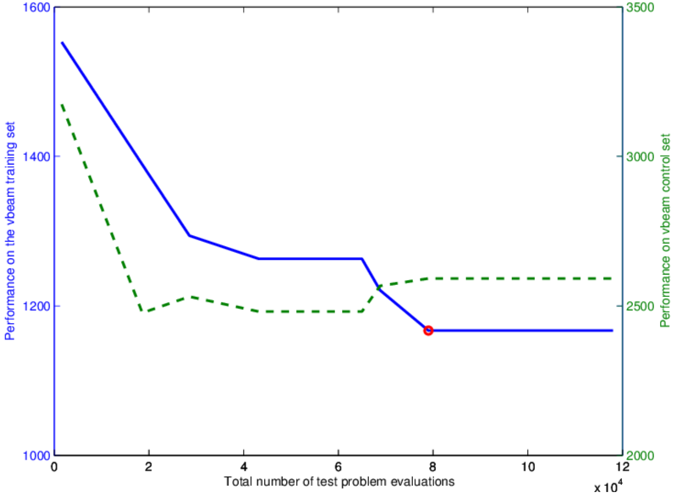
Does this work? Experiment on 2 specific classes of (minimization) problems

- nonlinear nonconvex trajectory tracking least-squares
- nonconvex regularized cubic models

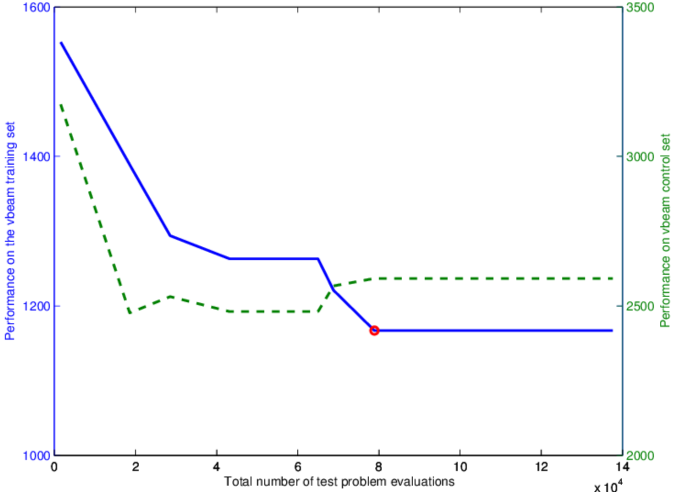
Trajectory tracking: AO training, medium-low error deviation, low accuracy



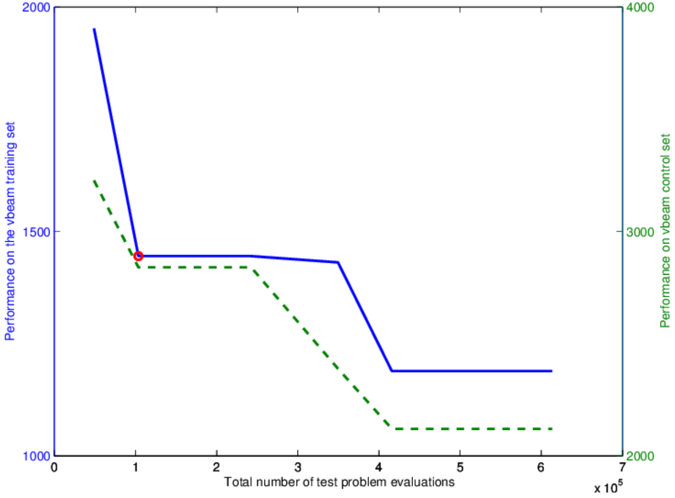
Trajectory tracking:
AO training, high error deviation, low accuracy



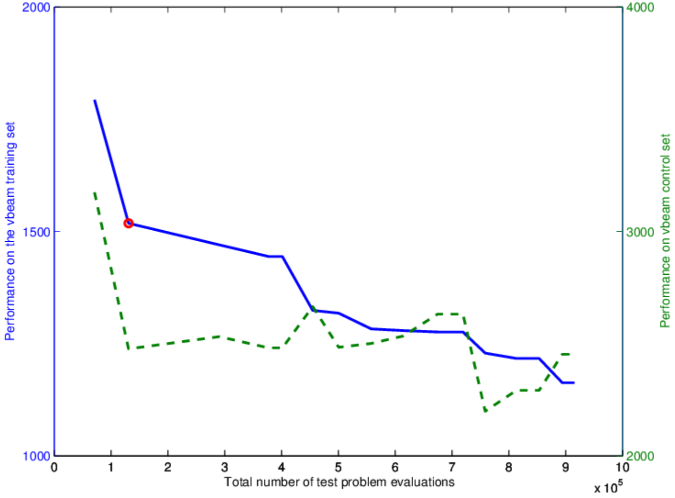
Trajectory tracking: AO training, high error deviation, high accuracy



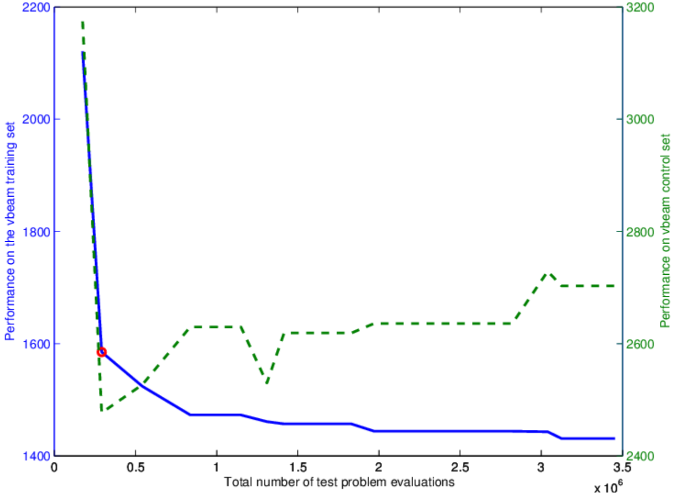
Trajectory tracking:
RO training, medium-low error deviation, low accuracy



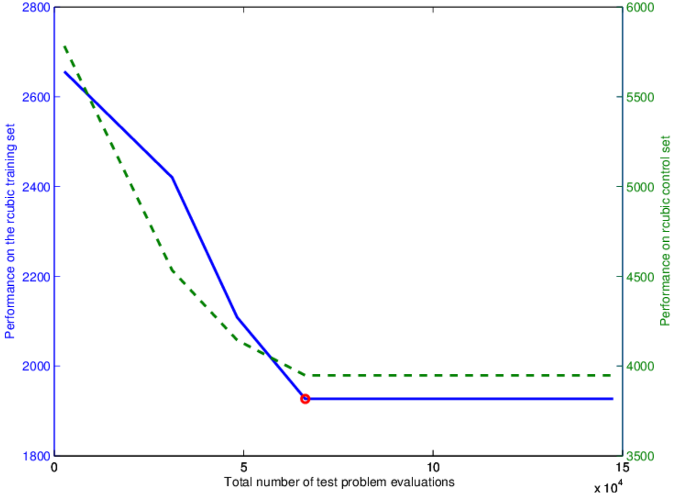
Trajectory tracking:
RO training, high error deviation, low accuracy



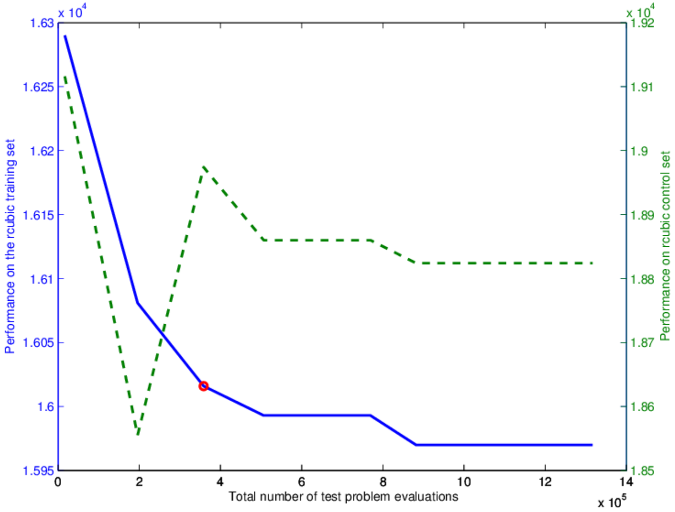
Trajectory tracking: RO training, high error deviation, high accuracy



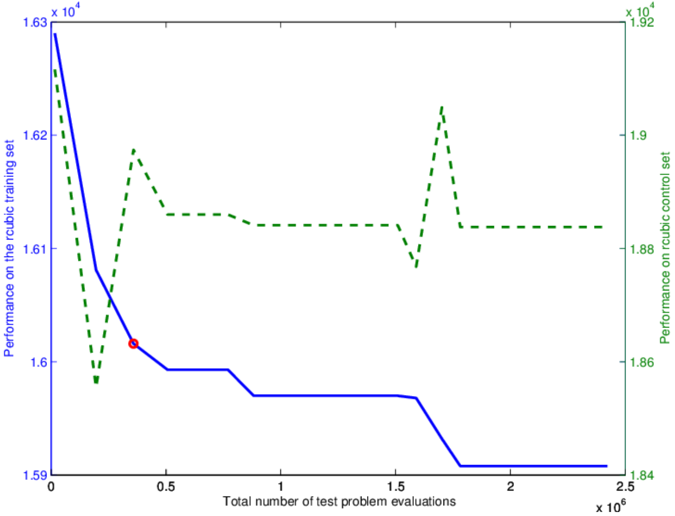
Regularized cubics: AO training, medium-low error deviation, low accuracy



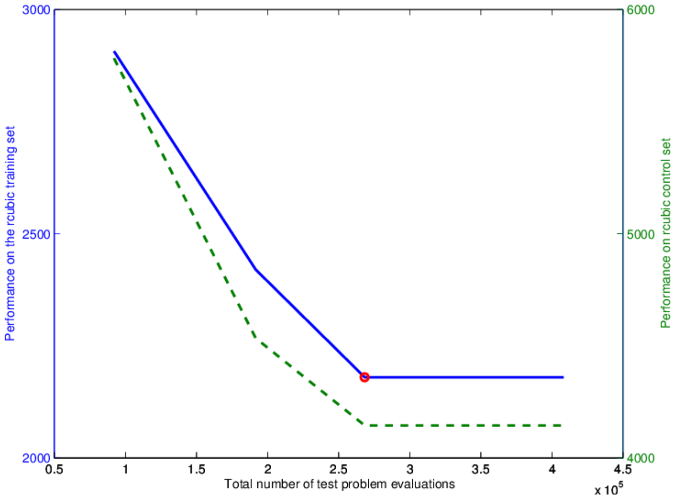
Regularized cubics: AO training, high error deviation, low accuracy



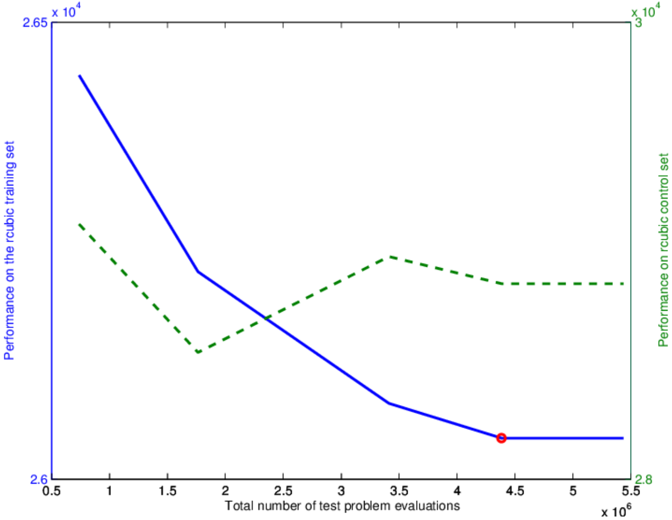
Regularized cubics: AO training, high error deviation, high accuracy



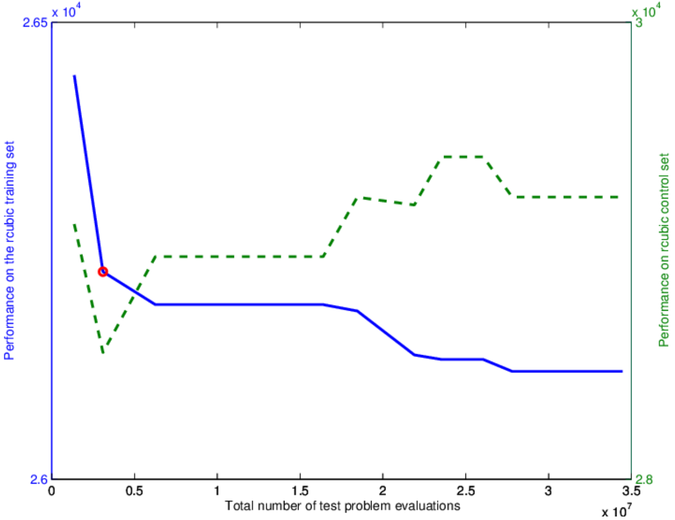
Regularized cubics:
RO training, medium-low error deviation, low accuracy



Regularized cubics: RO training, high error deviation, low accuracy



Regularized cubics: RO training, high error deviation, high accuracy



Additional BFO features

- MATLAB code (single file)
- very flexible interface
- optional user-defined variable's scaling
- incomplete function evaluations (crucial for training)
- checkpointing and restart
- flexible termination rules (including objective-function target)
- BFGS finish (for smooth problems)
- direct CUTEst interface

Examples of calls

- `[ x, fx ] = bfo( @banana, [ -1.2, 1 ] )`
- `[ x, fx ] = bfo( @banana, [ -1.2, 1 ], 'xtype', 'ic' )`
- `[ x, fx ] = bfo( @banana, [ -1.2, 1 ], 'xlower', 0, 'epsilon',0.01)`
- ```
  [ x, fx ] = bfo( @banana, [ -1.2, 1 ] , ...
                   'save-freq',10,'restart-file','bfo.rst')
  ```
- ```
  [ x, fx ] = bfo( @banana, [ -1.2, 1 ] , ...
                   'training-mode', 'train', ...
                   'training-parameters', 'fruity', ...
                   'training-problems', {@banana,@apple},...
                   'training-problems-data', {@fruit_data} )
  ```
- ```
  [ x, fx ] = bfo( @robust_training, [ 0, -1, 0, 1 ] , ...
                   'xlevel', [ 1 1 2 2 ], ...
                   'max-or-min', [ 'min', 'max'] )
  ```
  ```

Some conclusions

- $\boxed{\text{*** Use BFO to tune your algorithm! ***}}$

  (you can even tune BFO to tune your own algorithms)
- The future: more complicated constraints, ...
- $\boxed{\text{More user-tunable codes?}}$

$\boxed{\text{Many thanks for your attention!}}$