Optimizing structured problems without derivatives and other new developments in the BFO package

Philippe Toint (with Margherita Porcelli)

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

(philippe.toint@unamur.be)

NAOIV-2017, Muscat, January 2017

How BFO was born...

- working on an interpolation-based derivative free optimizer for

$$\min_{x \text{ subject to bounds}} f(x)$$

(more on that at the very end)

- needed something quick and dirty to improve its parameter settings
- wrote a "Brute Force" tool ...
- ... which (after some years of tweaking) has turned into $\boxed{\text{BFO}}$

$$\boxed{\text{simple ideas} + \text{some computing power} \overset{?}{\Longrightarrow} \text{robust/useful tool?}}$$

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? (1)

Assume that the (negative) performance perf(params, prob) can be measured by running the considered algorithm with parameters params on problem prob (ex: number of function evaluations).

- 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):

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

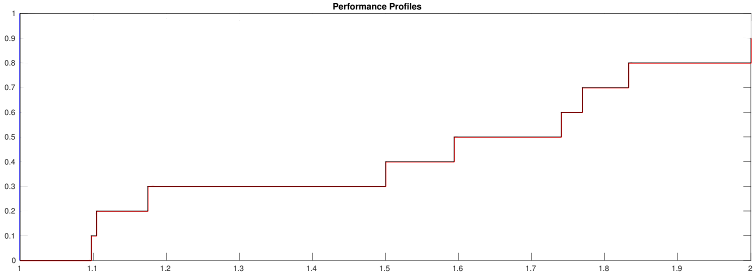
where

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

- Third model: optimize the performance profile !!!NEW!!!:

$$\pi_v(t) = \begin{array}{l}\text{proportion of problems solved by variant } v\\ \text{within } t \text{ times the performance of the best variant}\end{array}$$

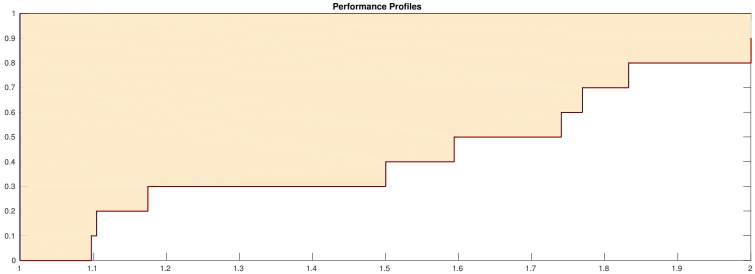
(fixed accuracy of $f(x_*)$)



- Third model: optimize the performance profile !!!NEW!!!:

$$\pi_v(t) = \begin{array}{l}\text{proportion of problems solved by variant } v\\ \text{within } t \text{ times the performance of the best variant}\end{array}$$

(fixed accuracy of $f(x_*)$)

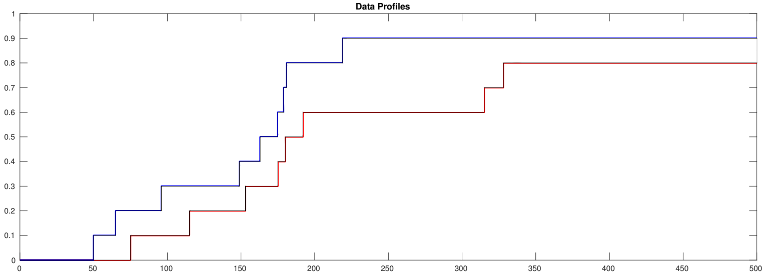


Which objective function? (4)

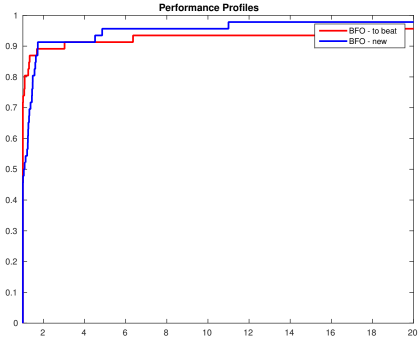
- Fourth model: optimize the data profile !!!NEW!!!:

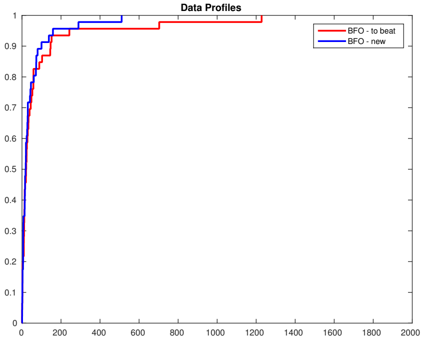
$$\delta_v(t) = \begin{array}{l}\text{proportion of problems solved by variant } v\\ \text{within a budget of } t \text{ evaluations}\end{array}$$

(fixed accuracy of $f(x_*)$)



Results for profile trainings



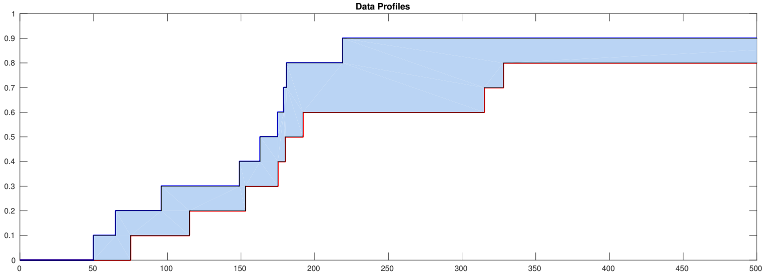


Which objective function? (5)

- Fourth model: optimize the data profile !!!NEW!!!:

$$\delta_v(t) = \begin{array}{l}\text{proportion of problems solved by variant } v \\ \text{within a budget of } t \text{ evaluations}\end{array}$$

(fixed accuracy of $f(x_*)$)



In practice: BFO (the Brute Force Optimizer)

$\boxed{\text{BFO}}$: 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/categorical variables
- handles multilevel/equilibrium problems
  (needed for the robust tuning strategy)
- includes self-tuning facilities

A simplified block view of BFO

Check inputs + initialize

↓

user-defined search step (optional)

↓

randomized poll step
(fw/bw moves along random directions)

↓

termination + mesh

= recursive (MIP, multilevel, training)

+ save/restore

Bound constraints, integer, lattice and categorical variables

Bound constraints

- detect which bounds are nearly active
- force their normals to belong to the set of poll directions
- include one-sided or truncated poll search

Integer or lattice variables

- align the initial grid with the integer grid
- avoid shrinking and rotations
- recursively explore a local tree of discrete subspaces
- keep track of record value in each such subspace to avoid re-exploration
- same thing if variables live on a user-specified lattice
- allows relaxation of integer variables !!!IN PROGRESS!!!

Categorical variables !!!NEW!!!

Additional algorithmic features:

- accumulate successful descent directions (exploiting "inertia")
- optional user-defined variables' scaling
- provision for multilevel optimization

$$\min_x \max_y \min_z f(x, y, z)$$

  with level-dependent bounds (equilibrium/game theory computations)
- incomplete function evaluations (crucial for training)
- flexible termination rules (including objective-function target)
- BFGS finish (for smooth problems)
- allows randomized termination test
- allows exploitation of problem structure !!!NEW!!!

Additional implementation features

- check-pointing at user-specified frequency
- allows objective functions with user-defined parameters
- very flexible keyword-based calling sequence
- MATLAB code (single file)
- direct CUTEst interface (for those interested)

Algorithmic parameters

User may specify (amongst others):

- grid shrinking/expansion factors
- inertia for defining progress directions
- initial scale in continuous variables
- local tree-search strategy (depth-first vs breadth-first)

(7 algorithmic parameters in total)

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?

BFO paper (TOMS) reports experiments on specific problem classes

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

+ very positive return from users

Exploiting problem structure (1) !!!NEW!!!

Consider coordinate partially-separable objective functions

$$f(x) = \sum_{i=1}^{p} f_i(x_i) \quad \text{where } x_i \text{ only involves a (small) subset of variables}$$

(very common, e.g, discretizations, block systems, . . .)

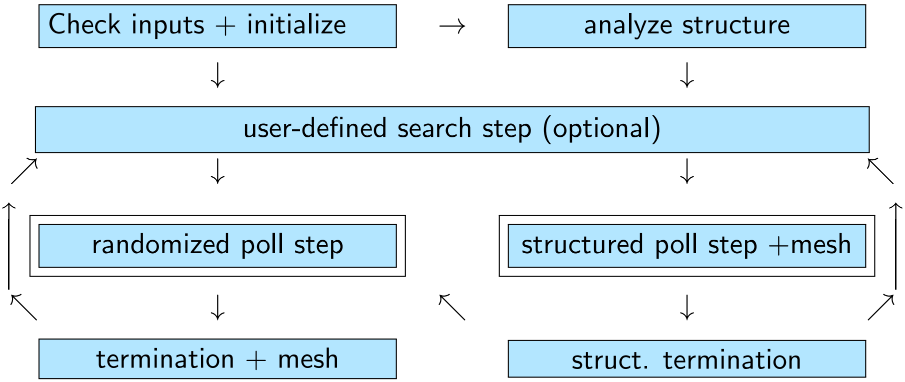
Key: no need to compute all f_i for moves along well-chosen coordinate-spanned subspaces!

$\Rightarrow$ allows parallel search along subspaces with independent f_i.

(Price & T., 2006)

$\Rightarrow$ allows independent mesh management within these subspaces

Exploiting problem structure (2) !!!NEW!!!



Exploiting problem structure (3) !!!NEW!!!

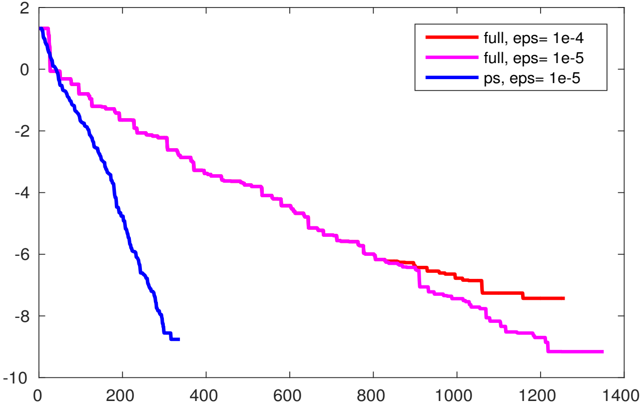
Extremely efficient to reduce the number of (full) evaluations

A few examples (ps/nops, ∞= '> 100000'):

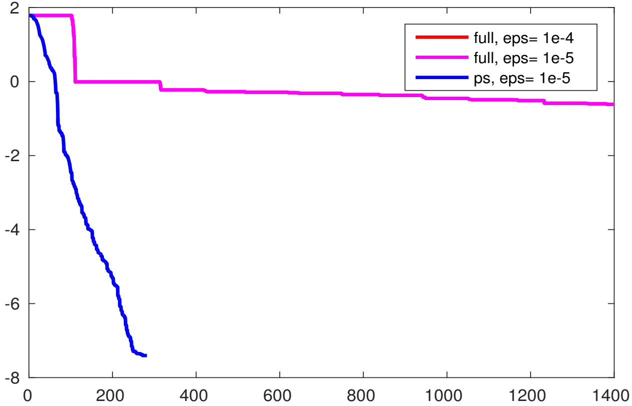
test probs	$n \approx 10$	50	100	500	1000	5000
ARWHEAD	133/967	402/13576	638/∞	2245/∞	3241/∞	10471/∞
BROYDEN3D	291/1298	256/23119	337/78777	398/∞	367/∞	989/∞
BROYDENBD	628/1325	1650/98458	1761/∞	2002/∞	1986/∞	2083/∞
CONTACT	—	222/29534	604/∞	895/∞	1814/∞	3620/∞
ENGVAL	166/1483	171/34466	183/∞	215/∞	253/∞	360/∞
DIXON7DGI	202/8455	249/∞	218/∞	430/∞	304/∞	500/∞
FREUDENROTH	419/2866	365/84351	154/∞	145/∞	178/∞	299/∞
HEL. VALLEY	128/2265	128/25529	158/∞	219/∞	348/∞	875/∞
MINSURF	—	393/21881	730/∞	1771/∞	3690/∞	8429/∞
NZF1	175/1899	217/∞	570/∞	649/∞	630/∞	776/∞
POWELL SING	554/26580	554/∞	604/∞	654/∞	654/∞	914/∞
ROSENBROCK	520/14270	707/∞	656/∞	1109/∞	1759/∞	4478/∞
TRIDIA	358/2440	293/∞	267/∞	353/∞	353/∞	505/∞
WOODS	1803/∞	1803/∞	1852/∞	1902/∞	2102/∞	2317/∞

Note: Most of the decrease in a number of evals independent of n
+ (relatively) slow checking for termination

Exploiting problem structure (4.1) !!!NEW!!!

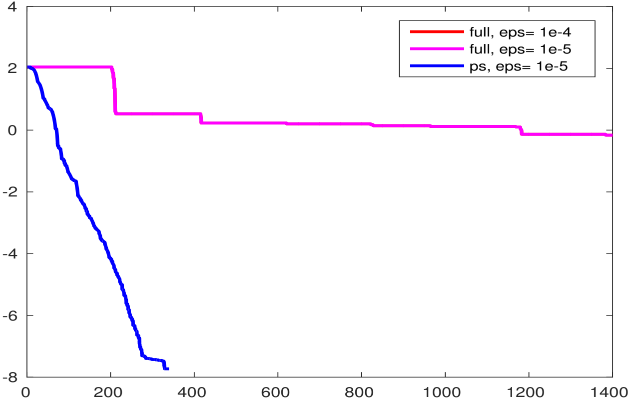


Broyden 3D, $n = 10$



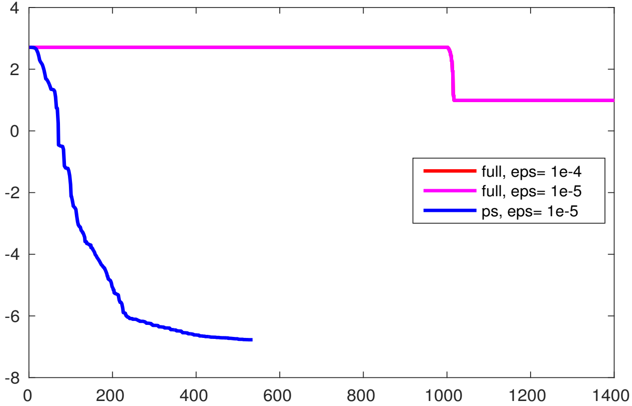
Broyden 3D, $n = 50$

Exploiting problem structure (4.3) !!!NEW!!!



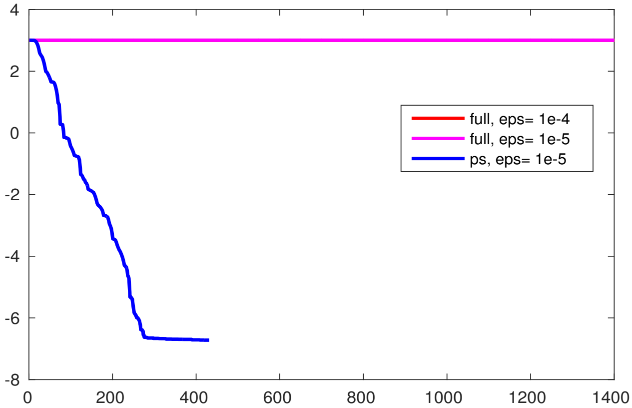
Broyden 3D, $n = 100$

Exploiting problem structure (4.4) !!!NEW!!!



Broyden 3D, $n = 500$

Exploiting problem structure (4.5) !!!NEW!!!



Broyden 3D, $n = 1000$

Categorical variables !!!NEW!!!

categorical variables =
variables whose values are user-defined (unordered) strings

- two types of (user-defined) neighbouhoods:
  - static: values are in a predefined list
    ex: {{ {'blue', 'black'}, '', {'blue', 'green', 'yellow'} }}
  - dynamic: values are defined on the fly by the user with possible changes in 'optimization context', i.e.
    - bounds
    - number of active variables (and hence objective function)

    extremely flexible, but burden of coherency on the user!
- except for neighbouhood's definition and relaxation, handled as integer variables

Does relaxing integer variables to continuous make the objective function undefined?

- (partially) solve the relaxed continuous problem
- find an integer (lattice) point close (in ℓ_1-norm) to the (approx) continuous solution (crash)
- use the result as starting point for full MIP optimization

Questions:

- perform relaxation at root node? every node? user-chosen nodes?
- relave accuracies of relaxed/unrelaxed optimization?
- handle unboundedness of the relaxed objective

For now: good results for root relaxation with low accuracy on the relaxed problem

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'] )
  ```
  ```

And now. . .

Some conclusions

*** Use BFO ***

*** Use BFO to tune your algorithm! ***

(you can even tune BFO to tune your own algorithms)

More user-tunable codes?

Perspectives for the BFO v 2.0 (somewhere in the spring)

- partially separable problems, categorical variables, profile training, relaxable MIPs, forcing function, search-step library, options file, . . .

Many thanks for your attention!

Further info. . .

Reading

M. Porcelli and Ph. L. Toint,
"BFO, a trainable derivative-free Brute Force Optimizer for nonlinear bound-constrained optimization and equilibrium computations with continuous and discrete variables",

TOMS, to appear, 2017

available from http://perso.unamur.be/˜phtoint/toint.html

Free download

Download BFO from the BFO site https://sites.google.com/site/bfocode/ !