New features of BFO Release 2.0

M. Porcelli* and Ph. L. Toint†

15 January 2020

Release 2.0 of the Matlab BFO package is a major upgrade from Release 1 and includes several important new problem-oriented features.

Using coordinate partially-separable problem structure. A large number of (often relatively large) optimization problems have some underlying structure, and BFO can be made to exploit this structure to considerable advantage.

The first case is when the objective function can be expressed as a sum of the form

$$f(x) = \sum_{i=1}^{p} f_i(x)$$

in which case we say that the objective function is in *sum form*. The functions $f_i(x)$ are the called *element functions*. A first advantage of problems in sum form lies in the possibilities for the user to define a structure exploiting search step function (for instance by building individual models for each of the element functions). Indeed, if the objective function is defined in sum form, BFO will automatically maintain an evaluation history for each individual element function, and will pass this information to the user when calling the user-defined search step (using BFOSS for instance).

The main advantage of problems whose objective function is in sum form is the specialization of this structure to the very important case where the problem is "coordinate partially separable" (CPS) or "sparse". In this extremely frequent case (e.g. when the objective function is related to a problems with distinct interconnected blocks, or is resulting from the discretization of a continuous problem), the objective function has the form

$$f(x) = \sum_{i=1}^{p} f_i(x_i),$$

*Università di Bologna, Dipartimento di Matematica, Piazza di Porta S. Donato,5, 40126, Bologna, Italy. Email: margherita.porcelli@unibo.it

†Namur Center for Complex Systems (NAXYS), University of Namur, 61, rue de Bruxelles, B-5000 Namur, Belgium. Email: philippe.toint@unamur.be

where now x_i only contains a subset of the problem's variables. A sum-form function is considered coordinate partially separable when the maximal dimension of one of the x_i is (often much) smaller than the total problem dimension. For example, the function

$$f([x(1)x(2), x(3)]) = \texttt{norm}([x(1)x(2)], 2)^2 + \texttt{norm}([x(2)x(3)], 'inf')$$

is coordinate partially separable with 2 elements of domains defined by the two vectors of variable indices [12] and [23]. Coordinate partially-separable functions as defined above are often called "sparse" because their Hessians (when they exist) are sparse matrices whose maximal dense principal submatrices are defined by the indices defining the x_i.

The use of any underlying coordinate partially separable structure results in very substantial gains in the number of evaluations of $f(x)$. Even if the use of this feature may not be critical for small problems, its use for problems of moderate or large size is often essential, especially if the cost of evaluating the element functions $f_i(x)$ is high. The gains in efficiency (evaluation counts) may typically be of several orders of magnitude (the amount of computation and storage internal to the algorithm therefore increases relatively to the number of evaluations). As demonstrated in [7], the evaluation gains can of course be combined with those resulting from a intelligent search-step step strategy due to the sum-form of the problem. The introduction and evaluation of this powerful technique is discussed in [7].

The BFOSS library of model-based search steps. As is common is pattern search optimization methods like BFO, the standard poll step can be completed by an often important *search step* (see [3], for instance). The idea behind such steps is to provide a surrogate model of the objective function in the neighbourhood of the current iterate x_{best}, which is built using information gathered in the course of the algorithm. This model can then be minimized (typically within some trust region) to provide an improved guess of the minimizer. Release 2.0 of BFO now also supplies BFOSS, a BFO-compatible library whose purpose is to compute such models and a resulting steps. BFOSS is based on interpolation models of various orders (from sublinear to fully quadratic) which interpolate available function values in a neighbourhood of the current iterate, thereby improving performance when compared to a standard poll-step algorithm.

In addition, BFOSS supports model building for objective functions given in sum-form and coordinate-partially-separable form (by building individual models for each element function). This combination provide even further significant performance improvements (as discussed in [7]).

Categorical variables. In addition to standard continuous and discrete variables, BFO now supports the use of categorical variables. Categorical variables are unconstrained non-numeric variables whose possible 'states' are defined by strings (such

as 'blue'). These states are not implicitly ordered, as would be the case for integer or continuous variables. As a consequence, the notion of neighbourhood of a categorical variable is entirely application-dependent, and has to be supplied, in one form of two possible forms, by the user. Moreover, the 'vector of variables' is no longer a standard numerical vector when categorical variables are present, but is itself a *vector state* defined by a value cell array of size n (the problem's dimension), whose i-th component is either a number when variable i is not categorical, or a string defining the current state of the i-th (categorical) variable. For example, such a 4-dimensional vector state can be given by the value cell array

{{ 'blue', 3.1416, 'green', 2 }}.

Variable i is declared to be categorical by specifying xtype(i) = 's'. If a problem contains at least one categorical variable, it is called a categorical problem and optimization is carried on vector states (instead of vectors of numerical variables). As a consequence, the starting point and the returned best minimizing point are vector states, and the objective function's value is computed at vector states (meaning that the argument of the function f is a vector state).

The user must specify the application-dependent neighbours (also called categorical neighbourhoods) of each given vector state with respect to its categorical variables. This can be done in two mutually exclusive ways.

1. The first is to specify *static neighbourhoods*. This is done by specifying, for each categorical variable, the complete list of its possible states. The neighbourhood of a given vector state vs wrt to categorical variable j (the hinge variable) then consists of all vector states that differ from vs only in the state of the j-th variable, which takes all possible values different from $vs(j)$. In this case, all variables of the problem retain their (initial) types and lower and upper bounds.
2. The second is to specify *dynamical neighbourhoods*. This more flexible technique is used by specifying a user-supplied function whose purpose is to compute the neighbours of the vector state vs 'on the fly', when needed by BFO. At variance with the static neighbouhood case, the variable number and types, as well as lower and upper bounds of the neighbouring vector states (collectively called the 'context') may be redefined within a framework defined by a few simple rules.

   In effect, this amounts to specifying the neighbouring nodes in a (possibly directed) graph whose nodes are identified by the list, types, bounds and values of the variables. As a consequence, the user-supplied definition of the neighbour(s) of one such node may need to take the values of all variables into account. Of course, for the problem to make sense, it is still required that the objective function can be computed for the new neighbouring vector states and that its value is meaningfully comparable to that at vs.

The very substantial flexibility allowed by this mechanism of course comes at the price of the user's full responsibility for overall coherence.

Performance and data profile training strategies. Because BFO is a *trainable package* (meaning that its internal algorithmic constants can be trained/optimized by the user to optimize its performance on a specific problem class), it needs to define training strategies which allow to decide if a particular option is better than another. Release 1 of BFO included the natural "average" training criterion (quality is measured by the average number of function evaluations on the class) and a robust variant of the same idea (see [5] for details). Release 2.0 now includes two new training strategies (introduced in [6]):

Performance profiling. When this training option is selected, the performance of two algorithmic variants (i.e. versions of BFO differing by the value of their internal algorithmic parameters) are compared using the well-known performance profile methodology [1, 2].

Data profiling. This option is similar to performance profiling, but uses data profiles [4] instead of performance profiles to compare two variants.

These new options correspond more closely to the manner in which derivative-free packages are compared in the optimization literature.

BFO Release 2.0 also improves performance and stability upon Release 1.0 and corrects a few bugs.

TRY IT OUT!

References

[1] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[2] E. D. Dolan, J. J. Moré, and T. S. Munson. Optimality measures for performance profiles. *SIAM Journal on Optimization*, 16(3):891–909, 2006.

[3] S. Le Digabel. Algorithm 909: NOMAD: Nonlinear optimization with the MADS algorithm. *ACM Transactions on Mathematical Software*, 37(4):1–44, 2011.

[4] J. J. Moré and S. M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization*, 20(1):172–191, 2009.

[5] 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. *ACM Transactions on Mathematical Software*, 44(1), 2017.

[6] M. Porcelli and Ph. L. Toint. A note on using performance and data profiles for training algorithms. *ACM Transactions on Mathematical Software*, 45(2), 2019.

[7] M. Porcelli and Ph. L. Toint. Global and local information in structured derivative free optimization with BFO. *arXiv:2001.04801*, 2020.