A note on using performance and data profiles for training algorithms

M. Porcelli* and Ph. L. Toint†

10 October 2018

Abstract

It is shown how to use the performance and data profiles benchmarking tools to improve the performance of algorithms. This paper proposed to achieve this goal by defining and approximately solving suitable optimizations problems involving the parameters of the algorithm under consideration. Because these problems do not have derivatives and may involve integer variables, we suggest to use a mixed-integer derivative-free optimizer for this task. A numerical illustration is presented (using the BFO package), which indicates that the obtained gains are potentially significant.

Keywords: algorithmic design, hyper-parameters optimization, trainable codes, derivative-free optimization.

Mathematics Subject Classification: 65K05, 90C56, 90C90.

1 Introduction

Making algorithms efficient and reliable is obviously desirable for both their designers and their users. Since most algorithms involve parameters, it is therefore important to choose them well. This is the purpose of hyper-parameters optimization (HPO) in machine learning that aims at finding a set of optimal parameters for a learning algorithm. Many approaches are available in this field such as grid or random search, Bayesian optimization, evolutionary methods and, finally, gradient-based strategies, see e.g. [4, 5] and references therein. The approach proposed in this paper is based on random-grid derivative-free optimization. In [11] the authors proposed BFO, a derivative-free optimization algorithm which is 'trainable', in the sense that it contains an internal procedure to select its algorithmic parameters to improve algorithmic performance, both from the point of view of the designer (using a large collection of diverse benchmarking cases) and of the user (focusing on a possibly more specific class of applications)(1). Obviously, improving performance requires a workable definition of this concept. As in [11], we assume that performance of an algorithm on a given problem can be measured by a number and that better performance corresponds to smaller such numbers. To make things concrete, and since

*Università degli Studi di Firenze, Dipartimento di Ingegneria Industriale, viale G.B. Morgagni 40, 50134, Firenze, Italy Email: margherita.porcelli@unifi.it

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

(1)This process is akin to the training of neural networks for pattern recognition, say, whose parameters are optimized to better identify known patterns.

we will be concerned below with derivative-free optimization, we shall consider from now on that performance is given by the number of objective function evaluations required by a solver to solve a given optimization problem[(2)]. Given a vector of algorithmic parameters, q, and a collection of benchmarking problems, $\mathcal{P}$, algorithmic performance was then measured using one of two 'training functions'. The first is the 'average' performance over all test problems [1, 2, 3], the second, inspired by robust optimization, is the average of the worst performance obtained for a slight variation of q. In both cases, the training process consisted in approximately optimizing those training functions as a function of the algorithmic parameters, under bound constraints on the admissible range for these parameters. It was shown in [11] that an approximate local minimization of either of these training functions can bring meaningful improvements in efficiency and reliability. The final comparison (and that with other derivative-free approaches) was then reported using the now widely accepted performance and data profiles techniques (see [7] for the first and [10] for the second).

The purpose of the present short note is to explain how it is possible to derive training functions from these two latter benchmarking measures, instead of merely using them for comparison. As in [11], we focus on the BFO derivative-free solver because it directly implements the relevant tools, but we stress that the approach is not limited to this particular case.

The paper is organized as follows. We first briefly recall, in Section 2, the definition of performance and data profiles given in [7, 10] and then derive the new training measures and associated training procedures in Section 3. A numerical illustration is reported in Section 4.

2 Performance and data profiles

Let $\mathcal{S}$ be a set of solvers (or solver variants) and let $\mathcal{P}$ be a set of benchmarking problems of cardinality $|\mathcal{P}|$. *Performance profiles* are defined in terms of a performance measure $t_{p,s} > 0$ obtained for each $p \in \mathcal{P}$ and $s \in \mathcal{S}$. We will consider here that $t_{p,s} > 0$ is the number of function evaluations required to satisfy a user-defined convergence test. For each $p \in \mathcal{P}$, let $\hat{t}_p = \min_{s\in\mathcal{S}} t_{p,s}$ and define $r_{p,s} = t_{p,s}/\hat{t}_p$ to be the performance ratio, so that the best solver s for a particular problem p attains the lower bound $r_{p,s} = 1$. We set $r_{p,s} = \infty$ when solver s fails to satisfy the convergence test on problem p. For $\tau \geq 1$, each solver $s \in \mathcal{S}$ and each problem $p \in \mathcal{P}$, one then defines

$$k(r_{p,s}, \tau) = \begin{cases} 1 & \text{if } r_{p,s} \leq \tau, \\ 0 & \text{otherwise.} \end{cases}$$

The performance profile for solver s is then given by the function

$$p_s(\tau) = \frac{1}{|\mathcal{P}|} \sum_{p\in\mathcal{P}} k(r_{p,s}, \tau), \quad \tau \geq 1.$$

By definition of $t_{p,s}$, $p_s(1)$ is the fraction of problems for which solver s performs the best, $p_s(2)$ gives the fraction of problems for which the solver's performance is within a factor of 2 of the best, and that for τ sufficiently large, $p_s(\tau)$ is the fraction of problems solved by s. More generally, $p_s(\tau)$ can be interpreted as the probability for solver $s \in S$ that the performance ratio $r_{p,s}$ is within a factor τ of the best possible ratio. Therefore, $p_s(1)$ measures efficiency of the solver while its robustness (high probability of success on the set $\mathcal{P}$) is measured in terms of $p_s(\infty)$. A key feature of performance profiles is that they give information on the *relative* performance of several solvers [6, 10], which therefore strongly depends of the considered set $\mathcal{S}$ of competing solvers or algorithmic variants [9].

[(2)] In other contexts, possible measures involve error on the result, computing time, memory usage, etc.

In order to provide a benchmarking tool that gives the behaviour of a solver independently of the other solvers in $\mathcal{S}$, Moré and Wild [10] proposed the *data profile* measure motivated by the user interest in the percentage of problems that can be solved with a certain computational "budget". For $\nu > 0$ and each $s \in \mathcal{S}, p \in \mathcal{P}$, one defines

$$g(t_{p,s}, \nu) = \begin{cases} 1 & \text{if } t_{p,s} \leq \nu(n_p + 1), \\ 0 & \text{otherwise,} \end{cases}$$

where n_p is the number of variables in $p \in \mathcal{P}$. The scaling by $n_p + 1$ is intended to consider the computational budget as in "simplex gradient" evaluations, rather than directly in function evaluations. The data profile for solver $s \in \mathcal{S}$ is then given by

$$d_s(\nu) = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} g(t_{p,s}, \nu), \quad \nu > 0,$$

and measures the percentage of problems that can be solved with ν "simplex gradient" evaluations.

3 New training measures and how to use them

We observe that, by definition, the plots of the performance and data profiles are staircase graphs and that, by the above discussion, the higher the curve corresponding to a solver, the better is its performance. This trivial observation suggests two new training strategies that simply consist in finding the parameter configuration that maximizes the area under the staircase graph generated by the performance or data profiles, respectively.

Let $\mathcal{Q}$ be the set of acceptable algorithmic parameters, $q \in \mathcal{Q}$ be a parameter configuration and let s_q be the solver variant with parameter configuration q. Consider data profiles first. We can define for each $q \in \mathcal{Q}$ the *data profile training function*

$$\phi^D_{\mathcal{P}}(q) \stackrel{\text{def}}{=} \int_{\nu_{\min}}^{\nu_{max}} d_{s_q}(\nu)\, d\nu,$$

where $0 \leq \nu_{\min} < \nu_{max}$ are user-specified values identifying a 'range of computational budgets' of interest, and then consider the corresponding data profile training problem

$$\max_{q \in \mathcal{Q}} \ \phi^D_{\mathcal{P}}(q). \tag{3.1}$$

The analogous problem for performance profiles is less obvious since, as discussed above, the computation of $p_{s_q}(\nu)$ depends on the behaviour of more than one solver, that is, in our case, on the performance of the trained solver with respect to different values of its algorithmic parameters q. We therefore propose to proceed sequentially from an initial parameter configuration q_0 and to evaluate the performance for a particular q by always comparing it to that obtained for q_0. Given the profile window $[\tau_{\min}, \tau_{max}]$ for some $1 \leq \tau_{\min} < \tau_{max}$ and the initial algorithmic configuration $q_0 \in \mathcal{Q}$, we define the *performance profile training function* $\phi^P_{\mathcal{P}}$ by

$$\phi^P_{\mathcal{P}}(q) \stackrel{\text{def}}{=} \int_{\tau_{\min}}^{\tau_{max}} \left[p_{s_q}(\tau) - p_{s_{q_0}}(\tau) \right] \, d\tau. \tag{3.2}$$

Training then corresponds to solving (possibly very approximately) the performance profile training problem

$$\max_{q \in \mathcal{Q}} \ \phi^P_{\mathcal{P}}(q). \tag{3.3}$$

In order to evaluate $\phi^P_{\mathcal{P}}$ and $\phi^D_{\mathcal{P}}$ in (3.1)-(3.3) respectively, one has to provide enough information to compute the profiles $p_s(\tau)$ and $d_s(\nu)$ during the training optimization process.

Let $q \in \mathcal{Q}$ be a parameter configuration and let s_q be the (BFO) algorithmic variant using parameters q. Let the profiles windows $[\tau_{min}, \tau_{max}]$ and $[\nu_{\min}, \nu_{max}]$ be given. We compare different parameter configurations declaring that the problem p with objective function f_p is solved by the variant s_q as soon as it produces an approximate solution x_q such that

$$f_p(x_q) \leq f_p^* + \chi(f_p(\bar{x}) - f_p^*) \stackrel{\text{def}}{=} c_p \tag{3.4}$$

where $\bar{x}$ is the starting point for the problem p, f_p^* is an approximation of the smallest obtainable value of f_p and $\chi \in [0,1]$ is a tolerance. The test (3.4) therefore compares the function value reduction $f_p(\bar{x}) - f_p(x_q)$ achieved by x_q relative to the best possible reduction $f_p(\bar{x}) - f_p^*$ [10]. We say that c_p, as defined in (3.4), is the *cut-off value* for problem p. Here, the iterates generated by the solvers are meant to be feasible with respect to any possible constraints present in the problem p.

Given an initial parameter configuration q_0, a starting point $\bar{x}$ and a tolerance $\chi > 0$, the training strategy proceed as follows. First, starting from $\bar{x}$, the solver variant s_{q_0} is run over the set $\mathcal{P}$ with high accuracy in order to evaluate the best objective found f_p^* for each $p \in \mathcal{P}$ and the resulting cut-off value c_p. Then, the number of function evaluations needed to the solver variant s_{q_0} to reach c_p, that is the value $t_{p,s_{q_0}}$, is retrieved. If data-profile training is considered, this is enough to compute the corresponding value of the objective $\phi^D_{\mathcal{P}}(q_0)$. The initial objective function value for performance-profile training is initialized to zero (see (3.2)). Optimizing the relevant objective function (i.e. (3.1) or (3.3)) can then be conducted (using BFO with its default parameters and its standard termination test in our case[(3)]), in the course of which the solver variant is run again with better and better values of the algorithmic parameters q, the performance measures t_{p,s_q} being always computed with respect to the initial cut-off value c_p. If for a certain problem p, a solver s_q fails to reach the value c_p within the prescribed number of f_p-evaluation, we follow the procedure recommended in [10]: its performance is set to a value which worse relatively to other solvers. It is worth noting that we did not find problems which could not be solved in any of the sample trials.

4 Numerical illustration

We now illustrate the above proposals by reporting some results obtained when training the BFO derivative-free optimization package by modifying its internal algorithmic parameters. We emphasize that the use of BFO is only meant to provide an illustration and not to discuss the merits of the package in this context, in particular relatively to other derivative-free solvers.

4.1 Experimental setup

BFO is a direct-search package for solving optimization problems in which the derivatives of the objective function are unavailable (for instance because the problem may involve integer variables). At a given iteration, it proceeds by sampling the values of the objective function at at points on a randomly oriented grid with adaptive mesh size (this process is called the poll-step). It then accepts an improved function value whenever it satisfies a 'sufficient decrease' condition relative to the current grid meshsize. If such an improvement is obtained, a new iterate is defined

[(3)] We are thus implicitly assuming that the results are relatively independent of the parameters and convergence test in BFO.

and the grid meshsize possibly increased. If this is not the case, the grid meshsize is decreased. As the iterations progress, a so-called 'inertia direction' is also computed using a number of past iterates and this direction is then privileged when constructing the grid. The minimization is terminated when the grid meshsize becomes smaller than a user-supplied threshold ϵ. Because the focus of this paper in not on the package itself, further inner details of the BFO method are of little interest here and we refer the interested reader to [11] for a full description. The BFO algorithmic parameters considered for training in our present experiments are presented in Table 4.1.

Parameters	q_0	l	u	Type	Description
α	1.5	1	2	c	The grid expansion factor
β	1/3	1/100	0.95	c	The grid shrinking factor
γ	5	1	10	c	The maximum grid expansion factor
δ	1	1/4	10^{-4}	c	The initial stepsize vector
η	1	1/4	10	c	The sufficient decrease fraction in the poll step
`inertia`	10	5	30	i	The number of iterations for the inertia direction

Table 4.1: BFO parameters selected for training with their initial configuration and bound constraints used in the training optimization problems.

We consider two set of benchmarking problems used in [11]. The first, named $\mathcal{P}_C$, consists in 55 bound-constrained problems with continuous variables of small dimensions extracted from the CUTEst library [8]. The list of problem names with their dimension is given in Table 4.2.

Name	n	Name	n	Name	n	Name	n	Name	n
ALLINIT	4	HADAMALS	4	HS38	4	MDHOLE	2	PENTDI	5
BDEXP	10	HARKERP2	10	HS3	2	NCVXBQP1	10	POWELLBC	6
BIGGSB1	10	HART6	6	HS3MOD	2	NCVXBQP2	10	PROBPENL	10
CAMEL6	2	HATFLDA	4	HS45	5	NCVXBQP3	10	PSPDOC	4
CHARDIS0	10	HATFLDB	4	HS4	2	NONSCOMP	10	QUDLIN	12
CHEBYQAD	4	HATFLDC	9	HS5	2	OSLBQP	8	S368	8
CVXBQP1	10	HIMMELP1	2	KOEBHELB	4	PALMER1A	6	SIMBQP	2
EG1	3	HS110	10	LINVERSE	9	PALMER2B	4	SINEALI	4
EXPLIN	12	HS1	2	LOGROS	2	PALMER3E	8	SPECAN	9
EXPLIN2	12	HS25	3	MAXLIKA	8	PALMER4A	6	WEEDS	3
EXPQUAD	12	HS2	2	MCCORMCK	10	PALMER4	4	YFIT	3

Table 4.2: The benchmark problem set $\mathcal{P}_C$: name and dimension n.

The second set, named $\mathcal{P}_V$, is made of nonlinear least-squares problems with bounds generated by varying the given data. The problem consists in fitting a nonlinear model of a vibrating beam to data by minimizing

$$f(x_1, x_2, x_3) = \sum_{j=0}^{16} \left[x_3 \tan\left(x_1 \left(1 - \frac{j}{16}\right) + x_2 \frac{j}{16} \right) - y_j \right]^2$$

where $x_3 \geq 0$. We fixed values for the three variables (by setting $x_1 = 0.21$, $x_2 = -0.35$ and $x_3 = 1$) and generated a class of 40 problems, where

$$y_j^1 = x_3 \tan\left(x_1 \left(1 - \frac{j}{16}\right) + x_2 \frac{j}{16} \right) \quad (j = 0, \ldots 16),$$

and

$$y_j^\ell = (1 + \eta^\ell) y_j^{\ell-1} \quad (j = 0, \ldots 16,\ \ell = 2, \ldots 40),$$

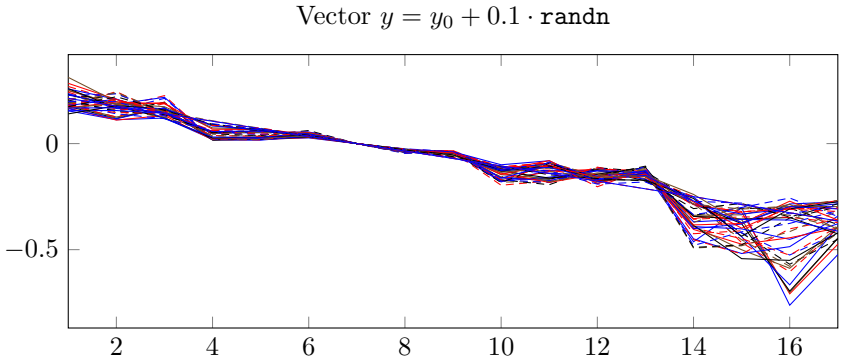

Figure 4.1: Set of vector data $\{y^\ell\}_l$ for the problems in $\mathcal{P}_V$.

with η^ℓ being a realisation of a Gaussian noise with zero mean and a prescribed value of the standard deviation σ. We set $\sigma = 0.1$ and obtained the vectors y plotted in Figure 4.1.

The solution f_*^p of each test problem is obtained (as is standard in data-profile experiments) by setting $\epsilon = 10^{-12}$ in the BFO convergence test and allowing 10000 function evaluations at most. We note that it is merely used to determine, once and for all, the cut-off value c_p associated with problem p. (It is *not* used for terminating the problem's solution during the algorithm training phase.) Starting from the initial parameter configuration q_0 in Table 4.1, the two optimization problems (3.1)-(3.3) are approximately, using the default version of BFO[(4)], solved imposing bound constraints on the parameters with bounds l and u reported in Table 4.1.

As in [11], we set the BFO termination threshold $\epsilon = 10^{-2}$ when solving the training minimization problems (3.1)-(3.3), as this value was found in [11] to give good results in terms of accuracy and avoid unnecessary overfitting. We also set an upper bound of 200 parameter configuration trials. Finally, the training is run using $\chi = 10^{-4}$ in (3.4). Experiments were carried out using Matlab R2016b on Intel Core i7 CPU 920 @ 2.67GHz x 8 12GB RAM.

Even if the training process using approximated minimizations of the relevant objective function guarantees improvements on the initial guess q_0, it is important to remember that there is absolutely no guarantee of reaching a local solution of the training problem, not to mention a global one. Thus the choice of q_0 may impact the final training result. However, our experience indicates that this impact is fairly limited.

4.2 Results for the CUTEst training set $\mathcal{P}_C$

We report in Table 4.3 the values of the trained BFO parameters obtained using the two training strategies on the set $\mathcal{P}_C$. Values of parameters are obtained using $\chi = 10^{-4}$ and setting the profile windows $[\nu_{\min}, \nu_{\max}] = [0, 2000]$ and $[\tau_{\min}, \tau_{\max}] = [1, 20]$, for the objectives in (3.1) and (3.3), respectively. Figure 4.2 clearly shows that the *performance improvements when using performance or data profiles are significant.*

We now discuss the effect on performance of varying the training windows. From the definitions, we would expect a profile window with small values (i.e. $\tau_{\max}$ relatively modest) to boost performance, while a window with larger values (substantial $\tau_{\max}$) to result in better reliability.

(4) Again, we stress that this choice is made for convenience, and is not meant to imply that BFO is the best possible method for the task. We also stress that both uses of BFO (for solving the training optimization problems and as the algorithm being trained) are completely independent: the version used for the first of these tasks is unaffected by the optimization conducted in the second.

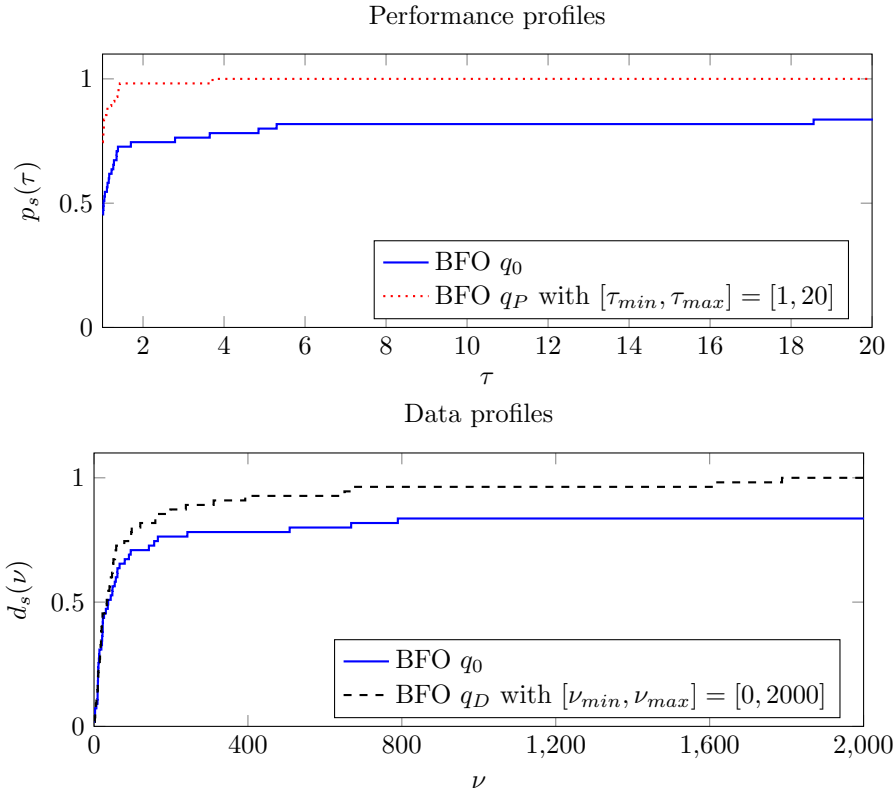

Figure 4.2: Set $\mathcal{P}_C$: performance profiles for BFO with default parameters q_0 against BFO with q_P (top), and data profiles of BFO with default parameters q_0 against BFO with parameters q_D (bottom). Parameters q_P and q_D are trained in the default intervals $[1, 20]$ and $[0, 2000]$, respectively.

	α	β	γ	δ	η	`inertia`
q_P	1.5	1/3	5.9	1	10^{-4}	11
q_D	1.6	0.32	4.8	0.25	10^{-4}	11

Table 4.3: Set $\mathcal{P}_C$: values of the trained parameters for the two training strategies.

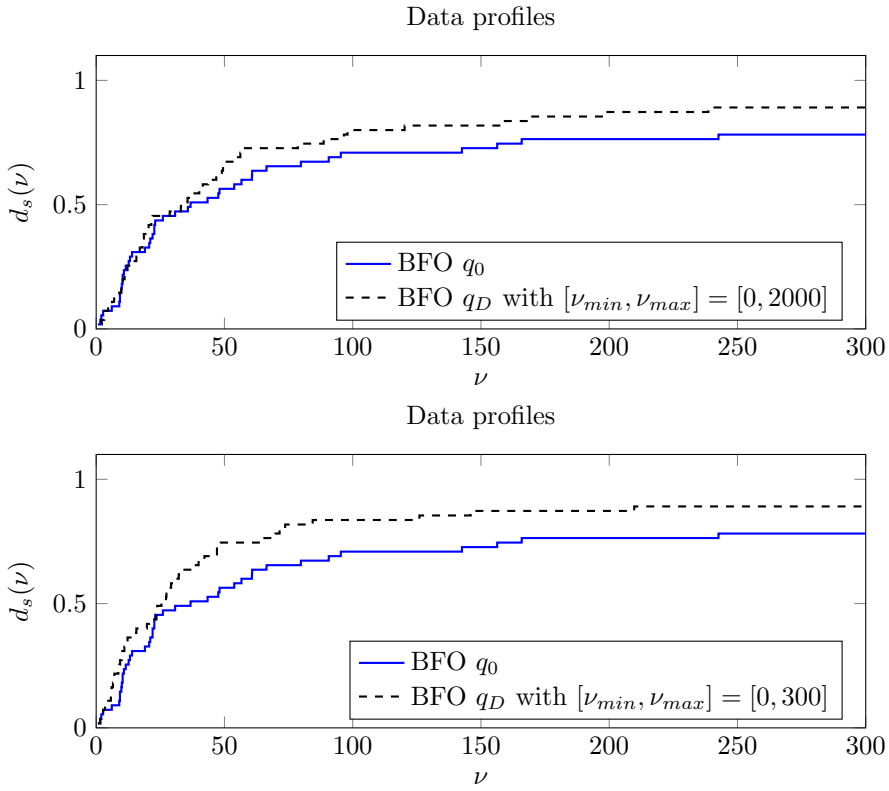

Figure 4.3: Set $\mathcal{P}_C$: zoom in the interval $[0, 300]$ of the data profiles obtained using BFO with q_0 and q_D trained in the default interval $[0, 2000]$ (top) and in the reduced interval $[0, 300]$ (bottom).

Because the performance profile result shows little room for improvement either in efficiency or reliability (as shown by Figure 4.2), we illustrate these effects (and their limits) using data-profile training.

We therefore repeated the training process using the data-profile objective function (3.1) from the same initial parameter configuration q_0, but using windows $[0, 300]$ and $[1500, 2000]$ instead of $[0, 2000]$. The resulting profiles are presented in Figures 4.3 and 4.4. While the expected improvement in efficiency using $[0, 300]$ is clearly visible in the first of these figures, the second shows that the procedure fails to produce an improved reliability when using the window $[1500, 2000]$, illustrating that approximately and locally minimizing the training function ($\phi_{\mathcal{P}}^D$ in this case) does indeed sometimes produce sub-optimal solutions.

4.3 Results for the vbeam set $\mathcal{P}_V$

Among the 40 generated vbeam problems, we used half of them as a training set and the other half

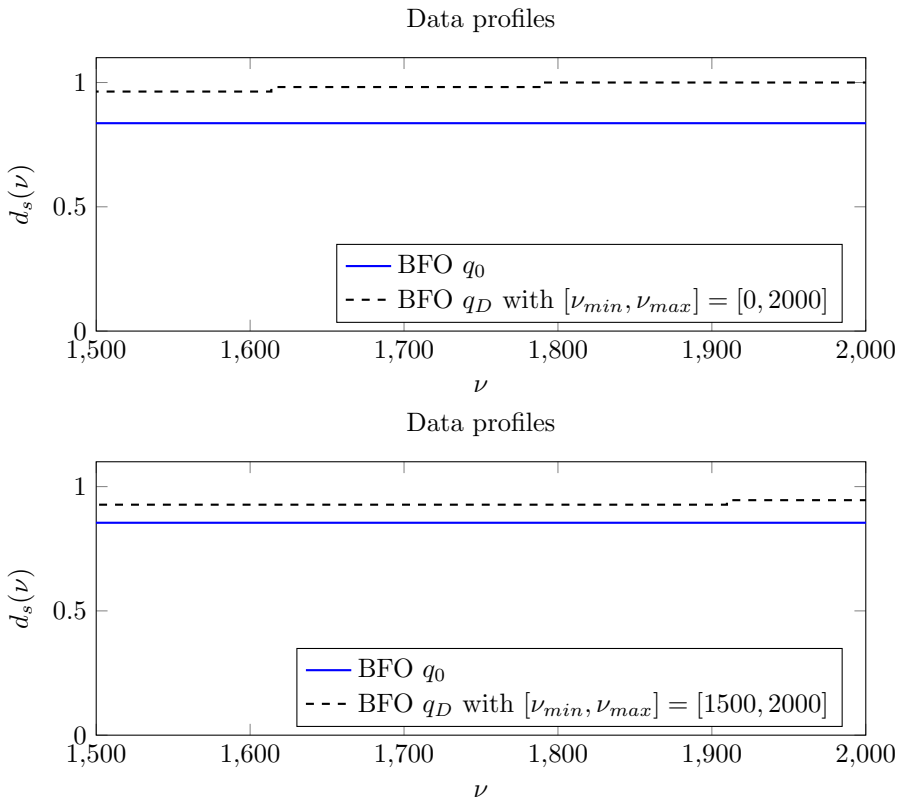

Figure 4.4: Set $\mathcal{P}_C$: zoom in the interval $[1500, 2000]$ of the data profiles obtained using BFO with q_0 and q_D trained in the default interval $[0, 2000]$ (top) and in the reduced interval $[1500, 2000]$ (bottom).

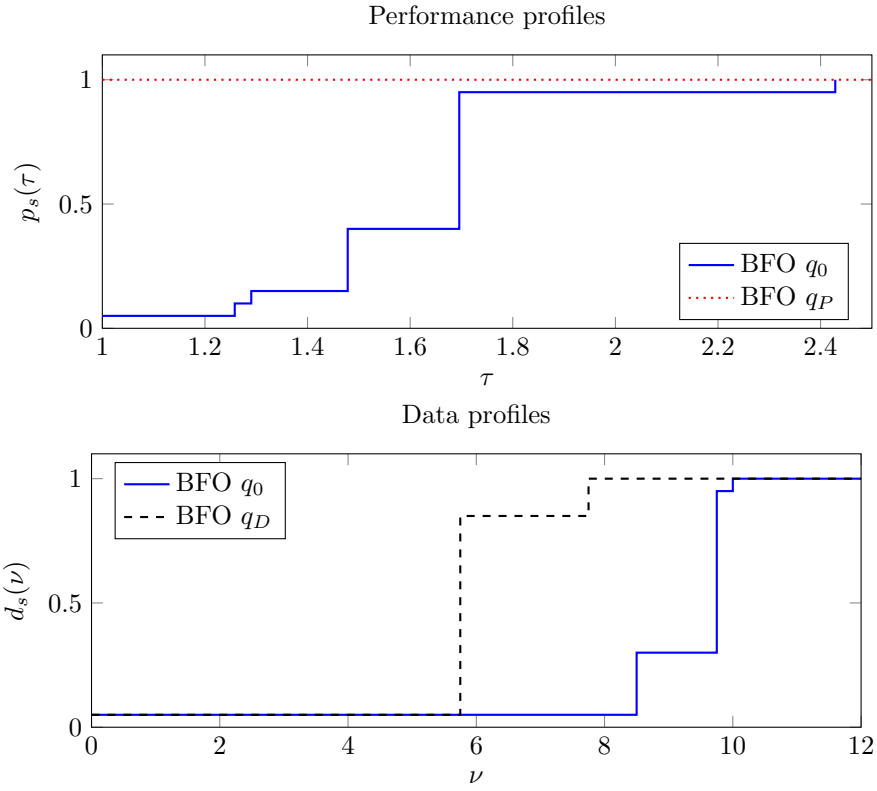

Figure 4.5: Set $\mathcal{P}_V$: performance (top) and data (bottom) profiles for BFO with different algorithmic parameters. Parameters q_P and q_D are trained in the intervals $[1, 20]$ and $[0, 2000]$, respectively.

as a control set. Results of the training phase are reported in Table 4.4 where we notice that the new training strategies yield the same trained parameters. Figure 4.5 shows clearly that q_P, q_D correspond to an "optimal" configuration in terms of performance profiles (the corresponding curves are horizontal lines with value 1). Also in terms of data profiles, the new strategy performs well.

	α	β	γ	δ	η	`inertia`
q_P, q_D	1.5	$7\,10^{-2}$	2.8	2.4	$1\,10^{-3}$	9

Table 4.4: Set $\mathcal{P}_V$: values of the trained parameters using different training strategies.

5 Conclusion

We have suggested how performance profiles and data profiles can be used to train algorithms and have illustrated our proposal by an application to the BFO package for derivative-free optimization. The results obtained show that significant gains in performance are possible but not guaranteed. The potential for improvement however suggests that the (careful) use of the proposed techniques is a useful tool in algorithmic design.

References

[1] C. Audet, C.-K. Dang, and D. Orban. Algorithmic parameter optimization of the DFO method with the OPAL framework. In K. Naono, K. Teranishi, J. Cavazos, and R. Suda, editors, *Software Automatic Tuning*, pages 255–274. Springer Verlag, Heidelberg, Berlin, New York, 2010.

[2] C. Audet, C.-K. Dang, and D. Orban. Optimization of algorithms with OPAL. *Mathematical Programming Computation*, pages 1–22, 2014.

[3] C. Audet and D. Orban. Finding optimal algorithmic parameters using derivative-free optimization. *SIAM Journal on Optimization*, 17(3), 2006.

[4] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, Boston, USA, 2011. MIT Press.

[5] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

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

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

[8] N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. *Computational Optimization and Applications*, 60(3):545–557, 2015.

[9] N. I. M. Gould and J. Scott. A note on performance profiles for benchmarking software. *ACM Transactions on Mathematical Software*, 43(2):1–15, 2016.

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

[11] 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.