

# Exploiting problem structure in derivative free optimization

M. Porcelli<sup>\*†</sup> and Ph. L. Toint<sup>‡</sup>

12 I 2021

## Abstract

A structured version of derivative-free random pattern search optimization algorithms is introduced which is able to exploit coordinate partially separable structure (typically associated with sparsity) often present in unconstrained and bound-constrained optimization problems. This technique improves performance by orders of magnitude and makes it possible to solve large problems that otherwise are totally intractable by other derivative-free methods. A library of interpolation-based modelling tools is also described, which can be associated to the structured or unstructured versions of the initial pattern search algorithm. The use of the library further enhances performance, especially when associated with structure. The significant gains in performance associated with these two techniques are illustrated using a new freely-available release of the BFO (Brute Force Optimizer) package firstly introduced in [42], which incorporates them. An interesting conclusion of the numerical results presented is that providing global structural information on a problem can result in significantly less evaluations of the objective function than attempting to building local Taylor-like models.

**Keywords:** derivative-free optimization, direct-search methods, structured problems, interpolation models.

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

## 1 Introduction

Derivative-free methods have enjoyed a continued popularity and attention from the very early days of numerical optimization [6, 30, 40, 52] to nowadays [3, 15, 32]. As their name indicates, such methods are aimed at solving nonlinear optimization problems without using derivatives of the objective function, which can be either too costly to compute, or simply unavailable (as is often the case in simulation-based applications). Among

---

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

<sup>†</sup>Institute of Information Science and Technologies “A. Faedo”, ISTI-CNR, Pisa, Italy.

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

all possible methods, we focus here on two broad classes: model-based techniques and pattern-search algorithms. In the first class, an explicit local model of the objective function is built, which is then used for finding any better approximation of the sought minimizer. Early proposals in this direction focused on unconstrained problems and include [17, 29, 36, 44, 45, 46, 47, 60], and were later analyzed and/or extended to the constrained case (see [7, 13, 14, 11, 16, 21, 48, 28, 55, 53, 51] among many others). The second class, also initially for unconstrained problems [1, 2, 6, 22, 30, 40, 41, 52], was later analyzed in [18, 19, 20, 23, 27, 31, 58, 59], and also extended to more general contexts (see [4, 33, 34, 42, 49, 35] for example).

Most of the contributions on derivative-free methods focus on small-scale unstructured problems. Indeed, building large-scale multivariate models is very expensive as it requires a number of data points (and costly function evaluations) which quickly grows with dimension: for the widely used quadratic polynomial models, this number grows like the square of the problem’s size. As a consequence, model-based approaches are in general unrealistic for moderate or large-scale applications. Similarly, the cost of the sampling strategies at the heart of pattern-search techniques also explodes with problem dimension and in practice restrict the application of these algorithms to optimization in very few variables (a few tens).

The situation fortunately improves if, instead of pure black-box optimization, one now considers the “gray box” case where one is allowed to exploit some underlying problem structure (while still avoiding the use of any derivative), see e.g. for an application [37]. Known sparsity in the objective function’s Hessian was considered in [9, 10] for methods based on quadratic models, where it was shown that the growth in function evaluations with size is, for a large class of sparse problems, essentially linear rather than quadratic. However, knowledge of the Hessian sparsity pattern is rarely directly obtained in practice without the knowledge of derivatives. What is much more common, for instance in discretized problems, is that optimization is performed on an application involving interconnected and possibly overlapping subsystems. While the detailed analytic expression of the subsystems’ models are often unavailable, it is not unusual for their connectivity pattern to be known. Such problems then often fall in the class of *coordinate partially separable* (or CPS) problems, where one attempts to solve

$$\min_{x \in \mathbb{R}^n} f(x) = \sum_{i=1}^q f_i(x) \quad (1.1)$$

where each  $f_i : \mathbb{R}^{n_i} \rightarrow \mathbb{R}$  is a possibly nonsmooth (or even non continuous) function depending on a subset  $\{x_i\}_{i \in \mathcal{X}_i}$  of the vector of variables  $x$ , for some index sets  $\mathcal{X}_i \subseteq \{1, \dots, n\}$  ( $i = 1, \dots, q$ ). The cardinality of these  $q$  subsets is denoted by  $n_i$  and is typically much smaller than the dimension  $n$  of  $x$ . In practice it is indeed common to see values of  $n_i$  of the order of ten or less, even for problems involving thousands of variables or more, see e.g. problems described in Section 4. Knowing the subsystems’ connectivity pattern then typically amounts to knowing the sets  $\mathcal{X}_i$  and being able to evaluate the “element functions”  $f_i$  individually. While we have introduced the concept for the unconstrained formulation (1.1), this is by no means restrictive, as constraints

may be added without affecting the structure. In the rest of this paper, we will assume that the problem’s variables are subject to simple bounds, that is

$$\ell \leq x \leq u, \tag{1.2}$$

where  $\ell$  and  $u$  are vectors in  $\mathbb{R}^n$  such that  $\ell \leq u$ , all inequalities being understood componentwise.

Optimizing coordinate partially separable problems<sup>(1)</sup> with model-based algorithms was considered in [11], where it was shown that the requirements in terms of function evaluations now depend on the maximum value of the  $n_i$ , which is often independent of the true problem’s dimension<sup>(2)</sup>. The use of the same problem structure was also initiated in [49], where a first pattern-search algorithm was described which could solve respectable size problems with  $n$  up to 5625.

The purpose of the present paper is to build on these contributions and to propose (in Section 2) a new derivative-free algorithm which is able to exploit the problem structure at pattern-search level (known as a “poll step”) as well as a library of partially separable multivariate polynomials to be combined with the structured poll step in a user-controlled “search step” (in Section 3). The dramatic impact of these features on numerical performance is then illustrated using a new version of the BFO (Brute Force Optimizer) package [42] incorporating them (in Section 4). In particular, *it is shown that the use of the global structural CPS information is often more efficient than that of local approximations of the objective function’s Taylor expansion.* It also shown that *a combined approach may also be advantageous*, at least for problems of moderate size. Some conclusions and perspectives are finally presented in Section 5.

## Notation

The  $j$ -th component of a vector  $x$  is denoted by  $x_j$ , while  $x_{\mathcal{I}}$  denotes the subvector defined by considering the components of  $x$  indexed by the index set  $\mathcal{I}$ .

## 2 Exploiting coordinate partial separability in the poll step

For the purpose of introducing our structure-exploiting techniques, we consider a generic pattern-search method consisting of a succession of iterations, each containing a search step (discussed in Section 3), a poll step, in which new (hopefully better) functions values are generated by taking steps along randomly generated orthogonal sets of search directions, and an update of the stepsizes as the iterations proceed. A description of such a simple pattern search for unconstrained problems is given in Algorithm 2.1. Comments on the constrained case are postponed to the end of this section.

---

<sup>(1)</sup>Coordinate partially separable problems are a subclass of the more general partially separable problems, where the objective function takes the form  $f(x) = \sum_{i=1}^q f_i(U_i x)$ , where the  $U_i$  are  $n \times n$  low-rank matrices.

<sup>(2)</sup>Coordinate separability has been extensively considered also in the context of derivative-based algorithms, see e.g. [50, 61], but, to our knowledge, in this setting the problem structure is assumed to be unknown and therefore is not exploited in the algorithm development.

**Algorithm 2.1: Outline of a simple pattern search algorithm**

**Initialization:** The initial iterate  $x$ , the initial stepsize  $\alpha$  and a convergence threshold  $\epsilon > 0$  are given. The parameters  $\gamma \geq 1, \beta, \eta \in (0, 1)$  are given. Define a set of orthonormal polling directions  $\{d^i\}_{i=1}^n$ .

**Until convergence**

**Search step:** Ask the user to provide a new (potentially improved) approximate minimizer of  $f$ , typically using problem specific modelling techniques;

**Poll step:** For  $j = 1, \dots, n$  or until “sufficient decrease” in  $f$  is obtained

- define a step of the form  $\alpha d^j$
- evaluate  $f(x + \alpha d^j)$  and, if necessary,  $f(x - \alpha d^j)$ ,

If  $f(x + \alpha d^j) < f(x)$  or  $f(x - \alpha d^j) < f(x)$ , replace  $x$  by  $x \pm \alpha d^j$ , depending on which step gave decrease;

Terminate the poll step if sufficient decrease is obtained, i.e. if

$$f(x \pm \alpha d^j) - f(x) < \eta \alpha^2. \quad (2.1)$$

**Termination step:**

- If  $f$  has decreased in the course of the poll step, increase the stepsize  $\alpha$  by setting  $\alpha \leftarrow \gamma \alpha$ , generate a new random set of orthonormal polling directions  $\{d^i\}_{i=1}^n$  (successful iteration) and start a new iteration;
- otherwise check for convergence (unsuccessful iteration): if

$$\alpha > \epsilon, \quad (2.2)$$

decrease  $\alpha$  by setting  $\alpha \leftarrow \beta \alpha$ , generate a new random set of orthonormal polling directions  $\{d^i\}_{i=1}^n$  and start a new iteration; otherwise declare convergence.

A few comments are useful at this stage. We first note that when sufficient decrease is obtained for one of the polling direction, as tested in (2.1), the algorithm stops using the current set of directions and directly updates the stepsize before starting a new iteration.

When (2.2) fails, that is when

$$\alpha \leq \epsilon, \tag{2.3}$$

it would be acceptable to terminate the optimization entirely. However, it is useful to continue the effort for finding a better point by applying a user-defined number of additional poll steps, each using a new random orthonormal basis  $\{d^i\}_{i=1}^n$ . If condition (2.3) is met at every such step, final convergence is then declared.

We assume, as indicated by (1.1), that all variables are continuous and that the polling directions are given by the canonical coordinate vectors  $\{e^i\}_{i=1}^n$ .

Two key observations allow to set up a strategy to exploit the partial separable structure in (1.1) that yields a new improved algorithm inspired by [49]. The first is that if a step along  $e^j$  is made, then only a subset of the element functions  $f_i$  will be affected and thereby need recomputation: only the  $f_i$  such that  $j$  appears in  $\mathcal{X}_i$  are concerned. Secondly, if variable  $x_k$  does not occur in any of the  $f_i$ 's involving variable  $x_j$ , a step of the form  $\alpha_j e^j + \alpha_k e^k$  ( $\alpha_j$  and  $\alpha_k$  are stepsizes) can be computed involving completely disjoint sets of element functions: those involving variable  $x_j$  and those involving variable  $x_k$ . Crucially, the cost of this combined step is potentially much *less* than an unstructured step along direction  $e^j$  where the complete  $f$  would be evaluated.

**Example.** *As an example, consider the partial separable function*

$$f(x_1, x_2, x_3, x_4, x_5) = f_1(x_1, x_2) + f_2(x_2, x_3) + f_3(x_1, x_2, x_4, x_5) + f_4(x_4, x_5) + f_5(x_4, x_5). \tag{2.4}$$

*Then, a step along  $e_1$ , say, only involves the element functions  $f_1$  and  $f_3$  while, since variable  $x_3$  does not occur in  $f_1$  and  $f_3$ , steps along  $\pm e_3$  can be made without affecting computations along  $e_1$ . We say that  $x_1$  and  $x_3$  belong to disjoint sets of variables. Remarkably, the evaluation at points of the form  $\alpha_1 e_1 + \alpha_3 e_3$  is less costly than a full function evaluation since it does not involve the evaluation of  $f_4$  and  $f_5$ .*

Exploiting these observations in a systematic way is the key to the new structured poll step. More specifically, once the user provides the sets  $\mathcal{X}_i$  ( $i = 1, \dots, q$ ) indicating the subset of  $\{1, \dots, n\}$  defined by the components of  $x$  which appear in the domain of the element function  $f_i$  ( $i = 1, \dots, q$ ), then this structure is automatically first pre-processed as described in Algorithm 2.2.

As in [49], it can be verified that, for each  $h \in \{1, \dots, t\}$ ,

$$\mathcal{M}_h \stackrel{\text{def}}{=} \bigcup_{k | \mathcal{I}_k \subseteq \mathcal{C}_h} \mathcal{Y}_k \subseteq \{1, \dots, q\}. \tag{2.5}$$

**Algorithm 2.2: Subspace structure detection**

1. The structure  $\{\mathcal{X}_i\}_{i=1}^q$  is first inverted in that sets  $\{\mathcal{E}_j\}_{j=1}^n$  are built such that

$$i \in \mathcal{E}_j \quad \text{if and only if} \quad j \in \mathcal{X}_i.$$

The sets  $\mathcal{E}_j$  contains the indices of all element functions  $f_i$  involving variable  $x_j$ .

2. Using lexicographic sorting, sets of variables' indeces  $\{\mathcal{I}_k\}_{k=1}^r$  corresponding to identical lists of elements  $\mathcal{E}_j$  are constructed, that is

$$j_1 \in \mathcal{I}_k \quad \text{and} \quad j_2 \in \mathcal{I}_k \quad \text{if and only if} \quad \mathcal{E}_{j_1} = \mathcal{E}_{j_2} \stackrel{\text{def}}{=} \mathcal{Y}_k.$$

The indeces of the element functions in each  $\mathcal{Y}_k$  are thus indistinguishable as far as their dependence of the problem's variables is concerned.

3. *Independent* collections  $\{\mathcal{C}_h\}_{h=1}^t$  of  $\mathcal{I}_k$  are then constructed by applying a greedy algorithm, independence being understood as the property that

$$\mathcal{I}_{k_1} \subseteq \mathcal{C}_h \quad \text{and} \quad \mathcal{I}_{k_2} \subseteq \mathcal{C}_h \quad \text{if and only if} \quad \mathcal{Y}_{k_1} \cap \mathcal{Y}_{k_2} = \emptyset.$$

Thus steps in variables belonging to different sets  $\mathcal{I}_k$  in  $\mathcal{C}_h$  can be computed independently (and in parallel). Moreover, this process only involves a subset  $\mathcal{M}_h$  of the set of all element functions. Overall, the following collections are generated as a result of the procedure described in Algorithm 2.2:  $\{\mathcal{X}_i\}_{i=1}^q$  and  $\{\mathcal{I}_k\}_{k=1}^r$  containing variable indeces,  $\{\mathcal{E}_j\}_{j=1}^n$  and  $\{\mathcal{M}_h\}_{h=1}^t$  containing element function indeces and  $\{\mathcal{C}_h\}_{h=1}^t$  containing indeces of sets of variables.

**Example.** *Returning to the the objective function described in (2.4), we have that  $q = 5, n = 5$  and*

$$\mathcal{X}_1 = \{1, 2\}, \quad \mathcal{X}_2 = \{2, 3\}, \quad \mathcal{X}_3 = \{1, 2, 4, 5\}, \quad \mathcal{X}_4 = \{4, 5\}, \quad \mathcal{X}_5 = \{4, 5\}$$

*and the corresponding sets of element functions' indeces defined in Step 1 of Algorithm 2.2 are given by*

$$\mathcal{E}_1 = \{1, 3\}, \quad \mathcal{E}_2 = \{1, 2, 3\}, \quad \mathcal{E}_3 = \{2\}, \quad \mathcal{E}_4 = \{3, 4, 5\}, \quad \mathcal{E}_5 = \{3, 4, 5\}.$$

*Step 2 aims at finding repeated lists of element functions, i.e.  $\mathcal{E}_4 = \mathcal{E}_5$  in this example, which are eliminated in the new sets of variables' and element functions' indeces. Therefore proceeding by lexicographic sorting, we get  $r = 4$  sets*

$$\mathcal{I}_1 = \{1\}, \quad \mathcal{I}_2 = \{2\}, \quad \mathcal{I}_3 = \{3\}, \quad \mathcal{I}_4 = \{4, 5\},$$

and

$$\mathcal{Y}_1 = \{1, 3\}, \mathcal{Y}_2 = \{1, 2, 3\}, \mathcal{Y}_3 = \{2\}, \mathcal{Y}_4 = \{3, 4, 5\}.$$

We then define the subspaces of independent variables using a simple greedy algorithm as follows. Initializing  $\mathcal{C}_1 = \{\mathcal{I}_1\}$ , we start checking if  $\mathcal{Y}_1 \cap \mathcal{Y}_2 = \emptyset$ . Since,  $\mathcal{Y}_1 \cap \mathcal{Y}_2 \neq \emptyset$  in our example, we now check if  $\mathcal{Y}_1 \cap \mathcal{Y}_3 = \emptyset$ . Since this is true,  $\mathcal{I}_3$  is added to  $\mathcal{C}_1$ . Then we check condition  $(\mathcal{Y}_1 \cup \mathcal{Y}_3) \cap \mathcal{Y}_4 = \emptyset$ . Since this condition fails, the definition of the set  $\mathcal{C}_1$  is completed and the next set  $\mathcal{C}_2$  is initialized as  $\mathcal{C}_2 = \{\mathcal{I}_2\}$ . Then one checks the intersection of  $\mathcal{Y}_2$  with  $\mathcal{Y}_4$  ( $\mathcal{I}_3$  has already been assigned). Since the intersection is not empty, we finally obtain  $t = 3$  collections of sets of independent variables

$$\mathcal{C}_1 = \{\mathcal{I}_1, \mathcal{I}_3\}, \mathcal{C}_2 = \{\mathcal{I}_2\}, \mathcal{C}_3 = \{\mathcal{I}_4\}. \quad (2.6)$$

The corresponding sets of element functions' indeces are defined by

$$\mathcal{M}_1 = \mathcal{Y}_1 \cup \mathcal{Y}_3 = \{1, 2, 3\}, \mathcal{M}_2 = \mathcal{Y}_2 = \{1, 2, 3\}, \mathcal{M}_3 = \mathcal{Y}_4 = \{3, 4, 5\},$$

In this examples, steps for variables in each collection  $\mathcal{C}_i$ ,  $i = 1, \dots, 3$  involves  $|\mathcal{M}_i| = 3$  element function evaluations.

Once the  $\{\mathcal{C}_h\}_{h=1}^t$  and  $\{\mathcal{M}_h\}_{h=1}^t$  are known, we may define the subspaces

$$\mathcal{S}_k \stackrel{\text{def}}{=} \text{span} \{e^j \mid j \in \mathcal{I}_k\}, \quad (2.7)$$

$x_{\mathcal{S}_k}$  the projection of the current iterate  $x$  onto  $\mathcal{S}_k$  and the ‘inactive’ index sets for the  $h$ -th collection as

$$\mathcal{N}_h = \{1, \dots, n\} \setminus \bigcup_{k \mid \mathcal{I}_k \subseteq \mathcal{C}_h} \mathcal{I}_k \quad \text{for } h \in \{1, \dots, t\}.$$

The structured poll step then consists in performing poll steps in the subspaces  $\mathcal{S}_k$  generated in each collection  $\mathcal{C}_h$  of sets of independent variables and checking the sufficient decrease condition (2.1) in  $f$  by only evaluating elements functions in the corresponding set  $\mathcal{M}_h$  (for  $k = 1, \dots, r$  and  $h = 1, \dots, t$ ).

Our updated algorithm using a structured poll step is then given by Algorithm 2.3.

**Algorithm 2.3: Pattern-search algorithm for partial separable problems**

**Initialization:** The initial  $x$ , the initial stepsize  $\alpha$ , a convergence threshold  $\epsilon > 0$  and the sets  $\mathcal{X}_i \subseteq \{1, \dots, n\}$  ( $i = 1, \dots, q$ ) specifying the connectivity pattern for problem (1.1) are given. The parameters  $\gamma, \iota \geq 1, \beta, \eta \in (0, 1)$  and  $n_2 < n$  are given.

**Structure analysis:** Compute a collection of sets of independent variables  $\{\mathcal{C}_h\}_{h=1}^t$  using Algorithm 2.2. Initialize the stepsizes  $\alpha_k = \alpha$  for each set  $\mathcal{I}_k, k = 1, \dots, r$  and define a set of orthonormal polling directions in  $\mathcal{S}_k, k = 1, \dots, r$  defined in (2.7).

**Until convergence**

**1. Search step:** Ask the user to provide a new (potentially improved) approximate minimizer of  $f$ , typically using problem specific modelling techniques;

**2. Structured poll step:** For  $h = 1, \dots, t$  or until “sufficient decrease” in  $f$  is obtained

**2.1 Poll step in  $\mathcal{C}_h$ :** For each  $k$  such that  $\mathcal{I}_k \subseteq \mathcal{C}_h$  perform a standard poll step (using random orthogonal directions in  $\mathcal{S}_k$  and stepsize  $\alpha_k$ ) starting from  $x_{\mathcal{S}_k}$  on the restricted function

$$f_{\mathcal{Y}_k} = \sum_{i \in \mathcal{Y}_k} f_i,$$

producing a potentially improved  $x_{\mathcal{S}_k}^+$  and decrease  $f_{\mathcal{Y}_k}(x_{\mathcal{S}_k}) - f_{\mathcal{Y}_k}(x_{\mathcal{S}_k}^+) \geq 0$ ;

If sufficient decrease in  $f_{\mathcal{Y}_k}$  is obtained, i.e. if  $f_{\mathcal{Y}_k}(x_{\mathcal{S}_k}^+) - f(x) < \eta\alpha_k^2$ , increase the stepsize  $\alpha_k$  by setting  $\alpha_k \leftarrow \gamma\alpha_k$ ; otherwise decrease  $\alpha_k$  by setting  $\alpha_k \leftarrow \beta^t\alpha_k$ . Generate a new set of orthonormal polling directions in  $\mathcal{S}_k$ .

**2.2 Iterate and function decrease update:** Define a new iterate  $x^+$  by

$$x_{\mathcal{I}_k}^+ = x_{\mathcal{S}_k}^+ \quad \text{for each } \mathcal{I}_k \subseteq \mathcal{C}_h$$

and

$$x_{\mathcal{N}_h}^+ = x_{\mathcal{N}_h} \quad \text{otherwise,}$$

and the corresponding objective function decrease by

$$f(x^+) - f(x) = \sum_{k | \mathcal{I}_k \subseteq \mathcal{C}_h} \left[ f_{\mathcal{Y}_k}(x_{\mathcal{S}_k}) - f_{\mathcal{Y}_k}(x_{\mathcal{S}_k}^+) \right].$$

If sufficient decrease in the objective function, i.e. if  $f(x^+) - f(x) < \eta\alpha_k^2$ , has been obtained, replace  $x$  by  $x^+$  and terminate the structured poll step.

**3. Termination step:** Update the value of the “global” stepsize  $\alpha = \min_k \alpha_k$ . If sufficient decrease in  $f$  was not obtained in the structured poll step and  $\alpha$  is not below the prescribed accuracy  $\epsilon$ , go to Step 1. Otherwise enter the second pass (Step 4).

**4. Second pass:** Generate a set of orthonormal polling directions  $\{d^i\}_{i=1}^{n_2}$  in the full space and compute a complete poll loop along these directions with stepsize  $\alpha$ . If sufficient reduction has been obtained, go to Step 1; else, declare convergence.

As is standard in a poll step, an objective function decrease may not be obtained in the structured poll step with the current choice of stepsize, in which case  $x_{\mathcal{S}_k}^+ = x_{\mathcal{S}_k}$  and  $f_{\mathcal{Y}_k}(x_{\mathcal{S}_k}) = f_{\mathcal{Y}_k}(x_{\mathcal{S}_k}^+)$ .

Once again we stress that using the successive collections  $\{\mathcal{C}_h\}_{h=1}^t$  is not mandatory (even if it clarifies the overall evaluation cost) and that poll steps can be performed on the subspaces  $\{\mathcal{S}_k\}$  using the restricted functions  $\{f_{\mathcal{Y}_k}\}$  completely independently and in parallel.

Compared with the method described in [49], the algorithm described in Algorithm 2.3 is both simpler and more efficient, as it performs complete poll steps using random orthogonal directions in each of the subspaces  $\mathcal{S}_k$  (instead of only retaining the best increment for steps along a fixed positive basis). Moreover, the mechanism used to adapt the stepsizes is more elaborate than the somewhat adhoc technique described in [49, Section 2.3]. It was also observed in practice that the stepsize reduction can be slightly faster than in the standard unstructured case. This faster reduction is translated in the algorithm by a stepsize shrinking factor  $\beta \in (0, 1)$  (used in the unstructured case) which is raised to some power  $\iota \geq 1$  (see Step 2.1 of Algorithm 2.3).

While the structured poll step is extremely efficient (as will be seen in the example below and in Section 4), it still has a potential drawback. Because the random poll directions are constrained to remain in each of the  $\mathcal{S}_k$ , they are not random directions in the complete space  $\mathbb{R}^n$ , and convergence is obtained, following [27], to points where no further decrease can be obtained in each of these subspaces. This unfortunately does *not* imply that no further decrease can be obtained for  $f$ . A second pass is therefore necessary for obtaining this desirable property (Step 4 in Algorithm 2.3). This second pass does not use structure and is therefore considerably less efficient. However, this is mitigated by the observation that most (and sometimes all) the decrease in objective function value is obtained in the first pass, the second pass often only playing the role of a (possibly mildly expensive) convergence check. Moreover, if sufficient decrease is identified during the second pass, a return is made to the structure-using mechanism of the first, in an attempt to efficiently improve the decrease obtained. Fortunately, the overall efficiency of the structured minimization remains, in all examples we have seen, orders of magnitude better than that of the unstructured one. Moreover, it is not unusual for applications of derivative-free algorithms that the user is above all interested in obtaining a significant decrease in the objective function and not so much in extracting the last carat of decrease, let alone in checking local optimality. In this case, the second unstructured pass may often be unnecessary, bringing the optimization cost further down.

In order to reduce the cost of the second pass, our implementation of Algorithm 2.3 allows the user to specify a small number  $n_2$  of random directions (typically much lower than  $n$ ) for the second pass.

We now discuss the cost of applying the structured poll step of Algorithm 2.3 compared with that of using Step 2 of Algorithm 2.1. Consider an unsuccessful poll step first, that is a poll step during which sufficient decrease is not obtained. (Note that such

steps must occur as the stepsizes have to become sufficiently small for the algorithm to terminate.) Because of (2.5), we see that, for a given  $h$ , the cost of evaluating the element function  $f_i$  for  $i \in \mathcal{M}_h$  once cannot exceed that of evaluating  $f$ , which we denote by  $c_f$ . As forward and backward moves are considered for every polling direction (at an unsuccessful poll step), the total evaluation cost of the unsuccessful structured poll step is at most  $2tc_f$ . By comparison, the cost of an unsuccessful unstructured poll step is equal to  $2nc_f$ . Since it is very often the case that  $t \ll n$  (see Table 4.1), *the evaluation cost of the structured step is typically only a small fraction of that of the unstructured one.* Because the poll step is terminated as soon as sufficient decrease is obtained, the cost of successful structured and unstructured poll steps is slightly more difficult to compare. As discussed in [27], the expected number of polling directions considered in a single successful (unstructured) poll step is small (typically 2 or 3). Two variants are however possible for the structured step. In principle, it can be terminated as soon as sufficient decrease is obtained on a given subspace  $\mathcal{S}_k$ . In the implementation discussed in Section 4, the loop on the subspaces associated to a collection  $\mathcal{C}_h$  is always completed before sufficient decrease triggers poll-step termination. This choice appears to be efficient and allows for parallel execution of the subspace-restricted poll steps for different subspaces. Its evaluation complexity therefore depends on the number of subspace collections  $\mathcal{C}_h$  examined (which is at most  $t$ ). If this number is also a very small integer (as is often the case), the evaluation cost of the structured poll step is very similar to that of the unstructured one. However, *the objective function decrease obtained is typically much larger*, as it corresponds to a number of unstructured successful poll steps given by the total number of subspaces  $\mathcal{S}_k$  considered in the calculation. Of course, further gains may be obtained in the structured case if the parallelism between the subspaces  $\mathcal{S}_k$  is also exploited.

When bound constraints are present, the backward and forward steps within the poll step are truncated to prevent computation of the objective function at infeasible points, but the rest of the calculation is essentially unchanged.

**Example.** *We now give a first taste of the effectiveness of the structured poll step. Consider an unconstrained problem with objective function structured as in (2.4) and whose element functions are given by*

$$f_1(x) = \sqrt{x_1^2 + x_2^2}, \tag{2.8}$$

$$f_2(x) = (\sin x_1 - 23x_2x_1)^2, \tag{2.9}$$

$$f_3(x) = (x_1^3 - 56x_2x_3)^2 - x_4, \tag{2.10}$$

$$f_4(x) = \max\{|x_4|, |x_5|\}, \tag{2.11}$$

$$f_5(x) = \sqrt{x_4^2 + x_5^2}. \tag{2.12}$$

*We observe that the full objective  $f$  is bounded below but the element  $f_3$  is unbounded in variable  $x_4$ . However, the collection of independent sets in (2.6) ensures that the*

restricted function  $f_{\mathcal{Y}_4}(x_{\mathcal{S}_4})$  is still bounded below when computing poll steps in the subspace  $S_4$  (the only one involving  $x_4$ ). In fact, the elements functions involved include  $f_3$  together with  $f_4$  and  $f_5$ . We then apply both the unstructured and structured variants of the method described above with a tolerance  $\epsilon$  on the stepsize equal to  $10^{-5}$ . Figure 2.1 shows the two resulting convergence histories in terms of the number of evaluations of the complete  $f$ . The advantage of using the structured poll step is striking.

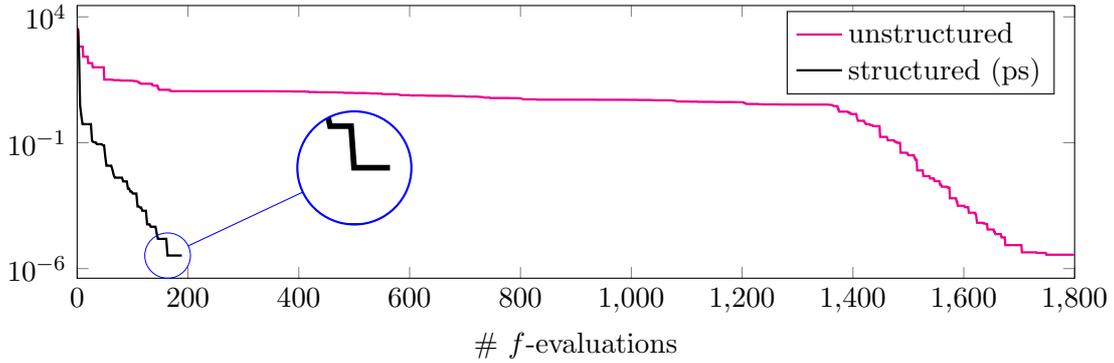


Figure 2.1: The evolution of  $f$  as a function of the number of (complete) evaluations for the objective defined in (2.8)-(2.12). Zoom on the function evaluations involved in the second pass when the partially separable structure is exploited (black curve).

The effect of the second pass is visible in the magnified part of the black curve (blue circle). The first flat segment indicates that the second pass is entered, but sufficient reduction is soon detected along a direction in the full space and the structured poll step is performed again (vertical segment). Then, the second pass is entered again (second flat segment) and convergence is finally declared.

### 3 A callable library of structured models

After discussing how to exploit the coordinate partially separable structure (1.1) in the poll step, we now briefly describe a multivariate interpolation technique that permits its exploitation in the user-controlled search step (i.e. Step 1 of Algorithms 2.1 and 2.3). As is common for such steps (see [33], for instance), the idea 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. We now describe a library named BFOSS for computing such a step while exploiting structure. As in [11, 53], it considers a surrogate model whose structure mirrors that of (1.1) in that it shares the same coordinate partially separable definition. This is achieved by constructing separate multivariate polynomial interpolation models for each of the  $f_i$  (restricted to their free variables). General multivariate polynomial

interpolation models follow the principles of [13, 55, 16], while their use in the context of partially separable problems is inspired by [11, 53] and the necessary adaptations used here for handling the bound constraints are similar to those discussed in [28]. While the general idea of structured models is not new, our experience suggests that its practical performance does depend on a number of more detailed decisions at the implementation level. We therefore focus, in what follows, on the algorithmic aspects which are specific to our approach and relevant to the numerical comparison to be conducted in Section 4. We first describe our strategy for the unstructured case ( $q = 1$ ) and then specialize it to the partially separable case ( $q \geq 1$ ).

Let  $Y = \{y^i\}_{i=1}^p$  be a sample set and let  $m(x)$  denote a polynomial of degree  $d$  ( $d = 1, 2$ ) interpolating  $f$  at the points in  $Y$ , that is satisfying the interpolation conditions

$$m(y^i) = f(y^i), \quad i = 1, \dots, p. \quad (3.1)$$

The polynomial  $m(x)$  can be expressed as a linear combination of elements of the natural basis  $\phi$  for the space of polynomial of degree at most 2, i.e. the basis of monomials, that we assumed to be ordered as follows

$$\phi = \left\{ 1, x_1, x_2, \dots, x_n, \frac{1}{2}x_1^2, \dots, \frac{1}{2}x_n^2, x_1x_2, \dots, x_{n-1}x_n, x_1x_3, \dots, x_{n-2}x_n, \dots, x_1x_n \right\}.$$

Finding a model that satisfies conditions (3.1) is equivalent to solving a potentially underdetermined linear system of the form

$$M_\phi z = f(Y) \quad (3.2)$$

with coefficient matrix  $M_\phi$  of dimension  $p \times \bar{p}$ ,  $z \in \mathbb{R}^{\bar{p}}$  and  $(f(Y))_i = f(y^i)$ ,  $i = 1, \dots, p$ , where  $p \leq \bar{p}$  and  $\bar{p}$  is the chosen number of elements from the basis  $\phi$ .

If the coefficient matrix  $M_\phi$  is square and nonsingular, then the model  $m$  is unique. If  $p < \bar{p}$  the linear system is underdetermined, the resulting interpolating polynomials may not exist or may no longer be unique, and different approaches to construct the model  $m(x)$  are possible [16]. BFOSS allows the user to choose between two possibilities: a model can be built by taking the minimum 2-norm solution of (3.2) [21] or, alternatively, a sub-basis  $\tilde{\phi}$  of  $p$  elements is extracted from  $\phi$  yielding the corresponding square matrix  $M_{\tilde{\phi}}$  (the components of  $z$  corresponding to the removed columns are set equal to zero).

As a result of the search step, a new tentative iterate  $x_+$  is then computed by minimizing the model  $m$  in the intersection between the trust-region and the feasible set, that is

$$\{x_{best} + s : \max(l - x_{best}, -\Delta) \leq s \leq \min(u - x_{best}, \Delta)\} \quad (3.3)$$

where  $\Delta > 0$  is the current trust-region radius, and  $f(x_+)$  is evaluated. Computing  $x_+$  within the trust-region (3.3) is a well-understood question and there are several algorithms available for the task (see [12, Chapter 7] for an overview). BFOSS allows the user to choose between a bound-constrained variant of the Moré-Sorensen trust-region algorithm and a projected gradient trust-region algorithm to solve (3.3) [38, 8, 57, 56].

The trust-region radius  $\Delta$  is updated on the basis of the ratio of the achieved to predicted reduction  $\rho = (f(x_+) - f(x_{best})) / (m(x_+) - m(x_{best}))$  as follows:

$$\Delta = \begin{cases} \min\{\alpha_1 \Delta, \Delta_{max}\} & \text{if } \rho > \eta_2 \\ \Delta & \text{if } \rho \in [\eta_1, \eta_2] \\ \max\{\epsilon_m, \alpha_2 \|s\|\} & \text{else} \end{cases}$$

where  $\alpha_1 > 1, \alpha_2 \in (0, 1), \Delta_{max} > 0$  and  $0 < \eta_1 < \eta_2 < 1$  are trust-region parameters [12, Chapter 17] and  $\epsilon_m$  denotes the machine precision. When BFOSS is called the first time in Step 1 of Algorithm 2.1, the value of  $\Delta$  is set equal to the current step size. At all subsequent calls, if  $\Delta$  is too small, i.e.  $\Delta < \Delta_{min}$  with  $\Delta_{min}$  defined by the user, the radius  $\Delta$  is restarted to the current step size. If  $\Delta$  is below  $\Delta_{min}$  after this restart, BFOSS is terminated. Finally, all the newly evaluated points ( $x_+$  and, when relevant, the new interpolation points) are returned to Algorithm 2.1 with their associated function values.

It is well-known that the fact that the model  $m$  is well-defined not only depends on the number of points in  $Y$ , but that a further geometric condition (known as *poisedness*) is also required (see [16] for details). We now describe the specific strategy used in BFOSS to define the interpolation set  $Y$ , which combines the use of models of various degrees and types and ensures the necessary poisedness condition.

When using BFOSS, the user selects a value of  $\bar{p}$  in the set  $\{n+1, 2n+1, (n+1)(n+2)/2\}$  corresponding to a linear model, a quadratic one with a diagonal Hessian and a full quadratic one. A first tentative set  $Y$  of  $\bar{p}$  points is built around  $x_{best}$  extracting from the recorded history (past points and function values), the closest points to  $x_{best}$  in the 2-norm and their associated function values (including  $x_{best}/f(x_{best})$ ). If the history is too short in that it only contains  $p < \bar{p}$  points, BFOSS uses all the available  $p$  points according to one of the strategy described above for the case  $p < \bar{p}$ .

Then matrix  $M_\phi$  in (3.2) is then built, together with its generalized inverse  $M_\phi^\dagger$ . The generalized inverse  $M_\phi^\dagger$  is constructed from a truncated SVD of  $M_\phi$ , where ‘‘redundant’’ singular values in a standard SVD are zeroed. We say that a singular value is redundant if its value is lower than the maximum singular value scaled by a user-defined parameter  $k_{ill}$  that measure the maximum ill-conditioning allowed. <sup>(3)</sup> Finally, the pseudoinverse is computed from this regularized SVD. When redundant singular values are detected, the corresponding points in  $Y$  are progressively removed, replaced by random points and  $M_\phi^\dagger$  rebuilt. Importantly, this crucial phase does not require the computation of additional function values.

Once  $M_\phi$  is considered sufficiently well-conditioned (within  $k_{ill}$ ) the poisedness of the current set  $Y$  is measured by computing the maximum absolute value of the Lagrange polynomials in the neighbourhood. Based on this measure, following [54], some points in  $Y$  can be replaced by some ‘‘far’’ points available in the history but not used so far. Then,  $Y$  is possibly further improved by performing exchanges until the improvement

---

<sup>(3)</sup>By default  $k_{ill} = \infty$  and  $k_{ill} = 10^{12}$  for structured and unstructured problems, respectively. These values resulted from our numerical experiments.

in poisedness becomes moderate enough. Finally, the obtained (reasonably conditioned)  $M_\phi^\dagger$  matrix is used to define the linear combination of the Lagrange polynomials which interpolates function values at the interpolation points.

The adaptation of this strategy to the partially separable case is straightforward: one simply applies the same technique to define interpolation models  $m_i$  for each of the element functions  $f_i(x_i)(i = 1, \dots, q)$ . Note that each model  $m_i$  has at most  $n_i$  variables and approximate  $f_i$ 's around the projection of  $x_{best}$  onto the subspace  $\mathbb{R}^{n_i}$ . The final trial point  $x_+$  is then computed by minimizing the global quadratic model

$$m(x_{best} + s) = \sum_{i=1}^q m_i(x_{best} + s),$$

in the box (3.3).

## 4 Numerical illustration

We now report numerical experiments comparing and combining the two techniques described above. The results were obtained by implementing the structured poll-step described in Algorithm 2.3 with the BFO (the Brute Force Optimizer) package [42] and exploiting the BFOSS library in combination with this upgraded version of BFO<sup>(4)</sup>. BFO is a random pattern search algorithm proposed by the authors in [42], whose structure is identical to that of Algorithm 2.1. Since a full description of this package is somewhat involved and many of its features irrelevant for our present discussion, we avoid restating it here in detail, and refer the reader to [42] for an in-depth description.

The main objectives of the numerical tests are

- to illustrate the impact of structure usage both at rather global level (via the use of the structured poll step) and a local level (by building local interpolation/regression models using BFOSS),
- to discuss the relative merits of the two techniques,
- to investigate the effectiveness of their combination.

We first provide some numerical illustration of our claim that the structured poll step is more efficient than the unstructured one. For this purpose, we compare the structured<sup>(5)</sup> and unstructured version of BFO<sup>(6)</sup> on a set of variable dimension test problems extracted from CUTEst [26] and/or already used in [49] for the most part.

---

<sup>(4)</sup>The BFOSS library is distinct from the BFO package itself –they come in different files– although it interacts with it. But its use or even presence is not necessary for running the main package.

<sup>(5)</sup>In the BFO implementation of the new poll step,  $\iota$  is a newly introduced algorithmic parameter which, like all such parameters [42], can be (and has been) trained for improved performance.

<sup>(6)</sup>It may be recalled that the unstructured BFO was shown in [42] to be quite competitive, in particular when compared with NOMAD [33].

For each problem, we considered dimensions ranging from around 10 to 10000, whenever solvable in reasonable time<sup>(7)</sup>. They are partitioned in four dimension-dependent test sets: small, smallish, medium and large. Variables are all considered to be continuous although the structured poll step is implemented in BFO for handling integer and categorical variables as well. The performance analysis with these variables are out of the scope of this work and will be considered in the future. Their main characteristics are detailed in Table 4.1, but we now give some additional information.

- The original BEALE problem from CUTEst only has two variables. The variant BEALES used here is obtained by juxtaposing  $n/2$  copies of the original problem, resulting in a totally separable problem (hence the S). It is useful as it exposes how well a method can exploit such an important structure.
- The function NZF1 is derived from that published in [49]. Its precise formulation is detailed in Appendix C. It reduces to the version of [49] for  $n = 13$ .
- The CONTACT problem is a bound-constrained minimum surface problem involving nonlinear surface boundaries and an obstacle from below the surface. It is described in Appendix B.
- The CUTEst minimum-surface problem NLINSRF only differs from LMINSRF (also in CUTEst) in that the surface boundary is nonlinear.
- The BROWNAL6 problem is that presented in [49] under the name 'Brown almost linear'.
- In the original MOREBV problem, the distance from the starting point to the solution decreases with dimension, which makes it less interesting for large problems. We use here a version of the problem where the original starting point is multiplied by  $\log_{10}(n)$  to compensate.
- The JNLBRNG1 is the journal bearing problems of MINPACK2 [5].
- Problems BROYDN3D, CONTACT, ENGVAl, JNLBRNG1, LMINSRF, NLINSRF, MOREBV are discretized problems. They illustrate how frequently the CPS structure appears in the situation: one variable associated to a particular location in the problem typically only depends on the variables associated with neighbouring locations, and not with all of them.

These problems are typical of a very large class of medium/large-scale applications, where, although  $q$  obviously depends on  $n$ , both  $t$  and  $\max_k |\mathcal{I}_k|$ , the maximal dimension of any subspace  $\mathcal{S}_k$ , do not.

Our experiments were conducted using a new release of BFO, which contains both the new structure-exploiting poll step and the BFOSS library. In order to train the

---

<sup>(7)</sup>Eight hours using Matlab R2017b on a Intel(R) Xeon(R) CPU E3-1245 v5 @ 3.50GHz machine with 64 GB RAM.

Problem	instance's dimensions ( $n$ )				$q$	$\max_k  \mathcal{E}_k $	$t$	$\max_k  \mathcal{I}_k $
	small	smallish	medium	large				
ARWHEAD	10	50, 100	500, 1000	5000, 10000	$n - 1$	2	2	1
BDARWHD	10	50, 100	500, 1000	5000, 10000	$n - 2$	3	3	1
BDEXP	10	50, 100	500, 1000	5000, 10000	$n - 2$	3	3	1
BDQRTIC	10	50, 100	500, 1000	5000, 10000	$n - 4$	5	5	1
BEALES	10	50, 100	500, 1000	5000, 10000	$n/2$	2	1	2
BROYDN3D	10	50, 100	500, 1000	5000, 10000	$n - 1$	3	3	1
BROWNAL6	10	50, 102	502, 1002	5002, 10002	$(n - 2)/4$	6	2	4
CONTACT	15	64, 144	400, 900	2500, 4900	$(\sqrt{n} - 1)^2$	4	4	1
ENGVAL	10	50, 100	500, 1000	5000, 10000	$n - 1$	2	2	1
DIXMAANA	15	51, 102	501, 1002	5001, 10002	$n$	4	4	1
DIXMAANI	15	51, 102	501, 1002	5001, 10002	$n$	4	6	1
FREUROTH	10	50, 100	500, 1000	5000, 10000	$n - 1$	2	2	1
HELIX	11	21, 101	501, 1001	5001, 10001	$(n - 1)/2$	3	2	2
JNLBRNG1	24	64, 144	400, 900	2500, 4900	$4(n_t + 1)^2$	3	3	1
LMINSURF	16	64, 144	400, 900	2500, 4900	$(n = 2(n_t + 2)(n_t + 1))$ $(\sqrt{n} - 1)^2$	4	4	1
MOREBV	12	52, 102	502, 1002	5002, 100002	$n$	3	3	1
NLMINSRF	16	64, 144	400, 900	2500, 4900	$(\sqrt{n} - 1)^2$	4	4	1
NZF1	13	39, 130	650, 1300	6500, 13000	$(7n/13) - 2$	6	4	2
POWSING	20	52, 100	500, 1000	5000, 10000	$n/4$	4	1	4
ROSENBR	10	50, 100	500, 1000	5000, 10000	$n/2$	2	1	2
TRIDIA	10	50, 100	500, 1000	5000, 10000	$n$	2	2	1
WOODS	20	40, 200	400, 2000	4000, 10000	$n/4$	4	1	4

Table 4.1: Characteristics of the test problem instances

parameter  $\iota$  defining the faster stepsize decrease in the structured optimization pass, we selected, for each of the above problems, the instance of third smallest dimension (mostly  $n \approx 100$ ) and performed training to improve the resulting data profile (see [43]). The experimental guess of  $\iota = 1.25$ , was only very marginally improved<sup>(8)</sup> to 1.2550. Note that other BFO algorithmic parameters were set to their default values. For small problems, the reported results are an average of 30 independent runs, for smallish 10 runs for medium ones, 5 runs and a single run for the large ones. All the results discussed in this section are reported in Table 1.3 in Appendix A.

### 4.1 Exploiting structure in the poll step

The results obtained by running the new structure-exploiting version of BFO (with the trained  $\iota$ ) are presented in Table 4.2. This table reports the number of complete function evaluation to obtain an approximate <sup>(9)</sup> minimizer for each of the instances of Table 4.1. For each instance, we give the number of required objective-function (full) evaluations for structure exploiting (first) and standard (second, no structure exploitation) versions

<sup>(8)</sup>We used the BFO default training accuracy requirement  $\epsilon_t = 0.01$ .

<sup>(9)</sup>We used the BFO default accuracy requirement  $\epsilon = 0.0001$ . We also stress that the algorithmic parameters in BFOSS have been trained for improved performance (as all other parameters of BFO) on the small test-set, using the data profile performance measure [43]. The values of the trained parameters are set as default in BFO and BFOSS.

Problem	small	smallish	medium	large			
ARWHEAD	79/962	91/11859	97/36085	146/∞	194/∞	389/∞	618/∞
BDARWHD	101/2152	70/∞	71/∞	81/∞	76/∞	77/∞	77/∞
BDEXP	1661/21122	13218/∞	34409/∞	∞/∞	∞/∞	∞/∞	∞/∞
BDQRTIC	290/2468	301/∞	298/∞	393/∞	542/∞	1150/∞	1480/∞
BEALES	275/∞	275/∞	275/∞	275/∞	275/∞	300/∞	325/∞
BROYDN3D	308/1225	199/22244	273/74758	304/∞	370/∞	640/∞	675/∞
BROWNAL6	15773/6171	4325/∞	2788/∞	2682/∞	2847/∞	3059/∞	3118/∞
CONTACT	265/268	221/30452	442/∞	1009/∞	1761/∞	3589/∞	5952/∞
DIXMAANA	185/2358	189/17357	240/55748	226/∞	375/∞	310/∞	756/∞
DIXMAANI	185/17702	186/∞	184/∞	230/∞	265/∞	709/∞	798/∞
ENGVAL	143/1567	155/30400	157/∞	159/∞	159/∞	159/∞	159/∞
FREUROTH	257/83101	181/∞	191/∞	185/∞	192/∞	233/∞	317/∞
HELIX	131/9931	152/∞	167/∞	298/∞	388/∞	883/∞	1070/∞
JNLBNG1	101/734	301/27567	427/∞	944/∞	1393/∞	1909/∞	1799/∞
LMINSURF	306/318	461/23692	1065/∞	3301/∞	8506/∞	29961/∞	∞/∞
MOREBV	7010/7154	60/22732	47/1923	47/9001	47/18001	47/90001	47/∞
NLMINSRF	412/415	579/29409	1082/∞	8853/∞	3412/∞	31619/∞	∞/∞
NZF1	177/1480	225/61772	625/∞	684/∞	667/∞	946/∞	2228/∞
POWSING	716/20605	824/∞	849/∞	988/∞	1036/∞	1164/∞	1148/∞
ROSENBR	361/13241	361/∞	384/∞	436/∞	461/∞	636/∞	736/∞
TRIDIA	440/3073	316/∞	314/∞	345/∞	293/∞	277/∞	278/∞
WOODS	1609/∞	1747/∞	1924/∞	2241/∞	2927/∞	4515/∞	5002/∞

Table 4.2: The numbers of objective function evaluations required by the structured/unstructured versions of BFO for the problem instances of Table 4.1, as a function of increasing problem size ( $\infty$  meaning that more than 100000 evaluations were needed).

of BFO <sup>(10)</sup>.

It clearly results from Table 4.2 that *it is possible to solve large partially-separable problems without using derivatives at an acceptable cost in number of function evaluations. Using the structure is crucial if the problem size exceeds ten or so.* In fact, a significant fraction of the problems of that size can't be solved at all in a reasonable number of evaluations if structure is neglected: direct derivative-free methods like BFO proceed by sampling, and this technique badly suffers from the curse of dimensionality.

A comparison of the above results with those of Table 2 in [49] also shows that the structured BFO significantly outperforms the simpler algorithm presented in that reference.

We now illustrate our comments of Section 2 about the relative efficiency of the two polling passes. Figures 4.2 to 4.4 show, for the **BROYDEN3D** problems in dimensions ten to one thousand, the objective function decreases obtained by the new BFO using structure (black line) and that obtained by the standard version of BFO which ignores structure (magenta line). These decreases are expressed as a function of the number of (complete) objective-function evaluations.

The rate of decrease of the structured BFO is very clearly much faster than that of

<sup>(10)</sup>When partially separable structure is provided, the number of full function evaluation per BFO iteration is retrieved by summing the number of element function evaluations and dividing by the number of elements (and rounding to integer).

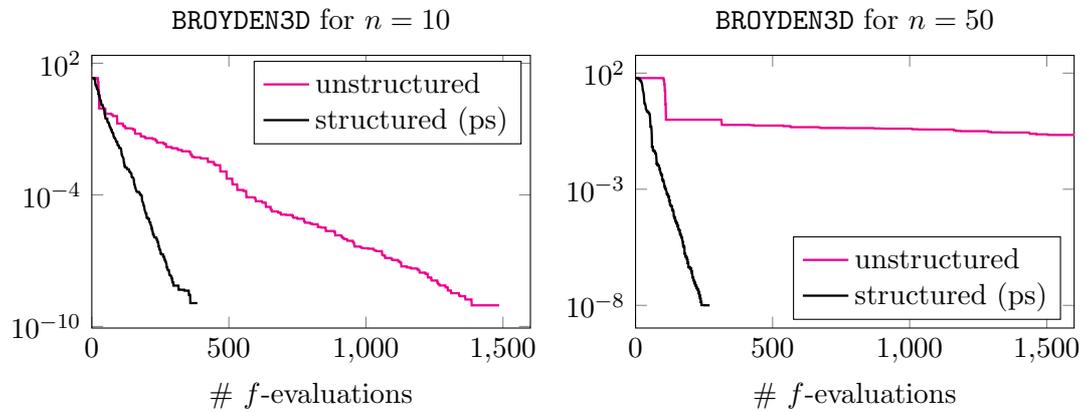


Figure 4.2: The evolution of  $f$  as a function of the number of (complete) evaluations for BROYDEN3D for  $n = 10$  and  $n = 50$

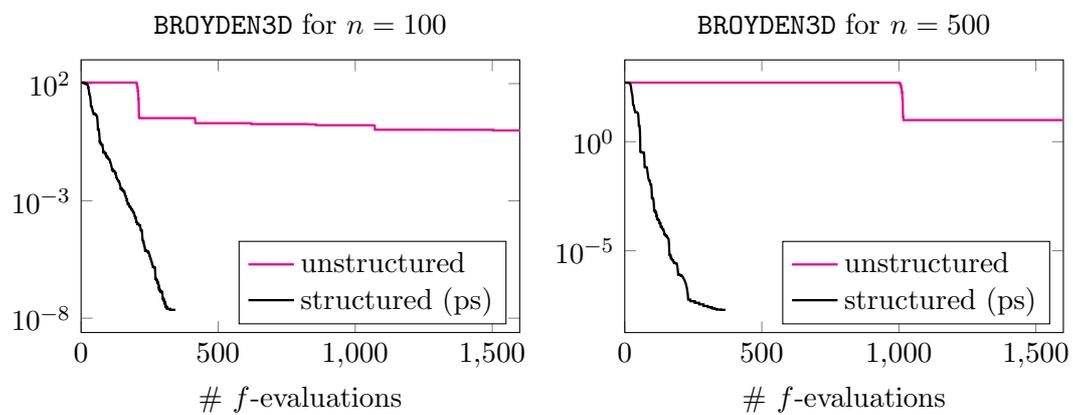


Figure 4.3: The evolution of  $f$  as a function of the number of (complete) evaluations for BROYDEN3D for  $n = 100$  and  $n = 500$

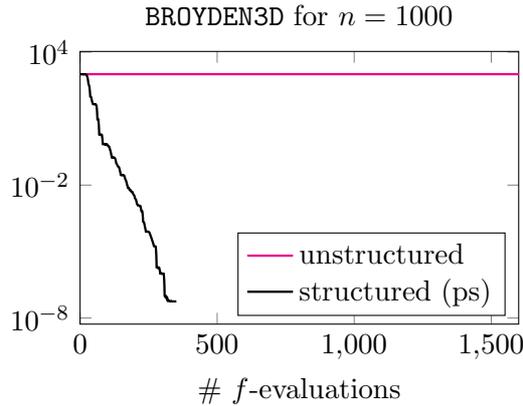


Figure 4.4: The evolution of  $f$  as a function of the number of (complete) evaluations for BROYDEN3D for  $n = 1000$

the unstructured variant. Moreover, the two passes of the structure-exploiting algorithm are very noticeable, the first pass being significantly faster and yielding most of the final decrease. We observe that the length of the second pass increases with size, which is expected because the space to sample in the second pass increases in dimension. However, because the first pass already made such good progress, the effort spent in the second pass remains acceptable, at least for the problem sizes considered here. By contrast, the performance of the standard unstructured version of BFO quickly degrades with size: for small dimensions, one notices the staircase-like decrease which is typical of pattern search methods. In this test problem, the first poll step is unsuccessful, leading to a full  $2n$  function evaluations before the stepsize is reduced. While this can be acceptable for small  $n$ , it becomes more problematic as  $n$  grows. For larger problems, the structured variant has already terminated before the first poll step is completed in the unstructured method. For the instance in 10 variables, one also notices (in the left picture of Figure 4.2) that it can be beneficial to reapply the structured poll-step mechanism of the first pass after a sufficient decrease in the second pass (cfr. Figure 2.1).

The behaviour of the structured variants on problem BDEXP finally merits a comment. This problem features a very flat objective function near the solution and, while the objective function value is decreased quickly, the mechanism of the pattern search method takes many iterations to declare optimality and terminate.

## 4.2 Exploiting structured models in the search step

We now turn to illustrating the performance which can be obtained using the BFOSS model library described above, and also compare it with the pure (structured and unstructured) sampling strategy of BFO alone. In the experiments reported next, we use fully quadratic models, the Moré-Sorensen method for maximizing the Lagrange polynomials in the trust-region and the truncated conjugate-gradient algorithm to solve (3.3).

In this section we use performance and data profiles [24, 39] to compare different

variants of BFO. For this purpose, we measure performance in terms of numbers of full objective function evaluations necessary for termination that is declared when the following condition holds

$$f(x_0) - f(x) \geq (1 - \tau)(f(x_0) - f_*). \quad (4.1)$$

Here  $x_0$  is the starting point for the problem,  $x$  is the solution returned by a solver,  $f_*$  is computed for each problem as the smallest value of  $f$  obtained by any solver within a given number  $\mu_f$  ( $\mu_f = 100000$  in our tests) of function evaluations,  $\tau \in [0, 1]$  is a tolerance that represents the percentage decrease from the starting value  $f(x_0)$  (we used the standard value  $\tau = 10^{-4}$ ). The stopping criterion (4.1) is suggested in [39] to generate profiles and differs from the default stopping criterion on the minimum step size used in BFO by default.

In Figure 4.5 four variants of BFO are compared on the small test. These variants are

**unstructured:** the standard unstructured BFO algorithm without using the BFOSS models,

**models:** the standard unstructured BFO algorithm where a full-dimensional BFOSS search step is attempted at every iteration,

**ps:** the version of BFO using the coordinate partially-separable structure, but without using the BFOSS models,

**ps & models:** the version of BFO using the coordinate partially-separable structure, using BFOSS models for each element function at every iteration.

This figure shows that, for small problems, the combined use of models and structure is the best algorithmic choice, but also indicates that using structure without models is clearly preferable to using models without structure. An interpretation of this observation is that providing global information on the problem (structure) outperforms approximating local one (models). The situation is less clear when the size of the problems increases, as is shown in Figures 4.6 and 4.7, where one compares the performance of the 'ps' and 'ps & models' variants on the smallish and medium test sets (the two other variants fail for a large proportion of the smallish problems). Profiles on the large test-set have not been generated since 'ps' was the only variant that could solve the test set in the maximum time allowed (8 hours).

In Figures 4.6 and 4.7, we see that the relative advantage obtained by the use of models for small problems progressively vanishes to disappear completely when the problem size grows.

While comparing the number of function evaluations, as we have done above, is most natural for derivative-free problems (the cost of an evaluation in real world applications often dominating that of all computations internal to the algorithm), it is also interesting to consider memory usage and internal computing effort. The pure unstructured variant is clearly the most economical from both points of view (but at the price of being the

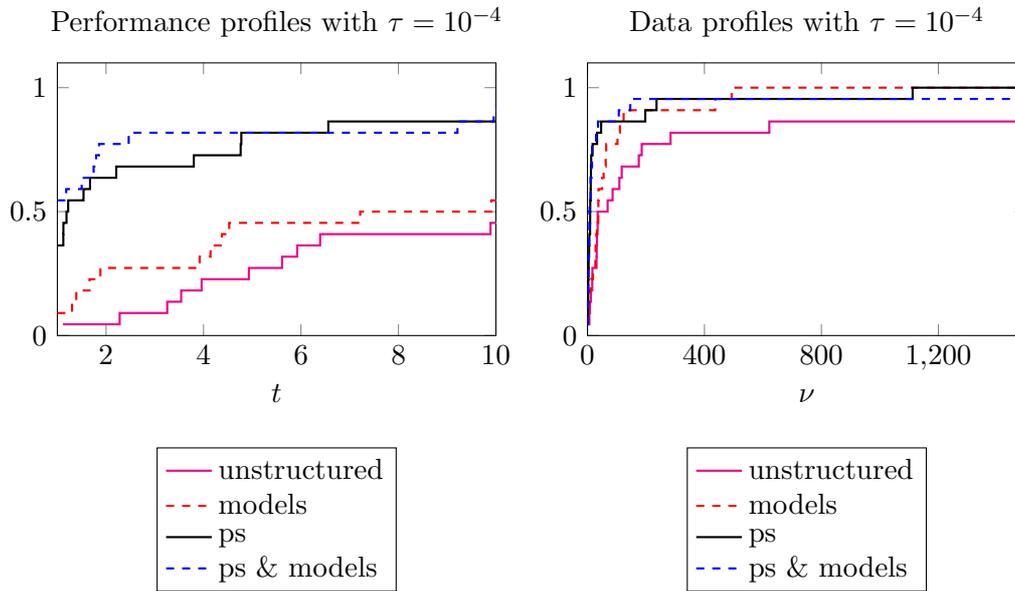


Figure 4.5: Performance and data profiles for unstructured/structured variants, with or without models (small test set).

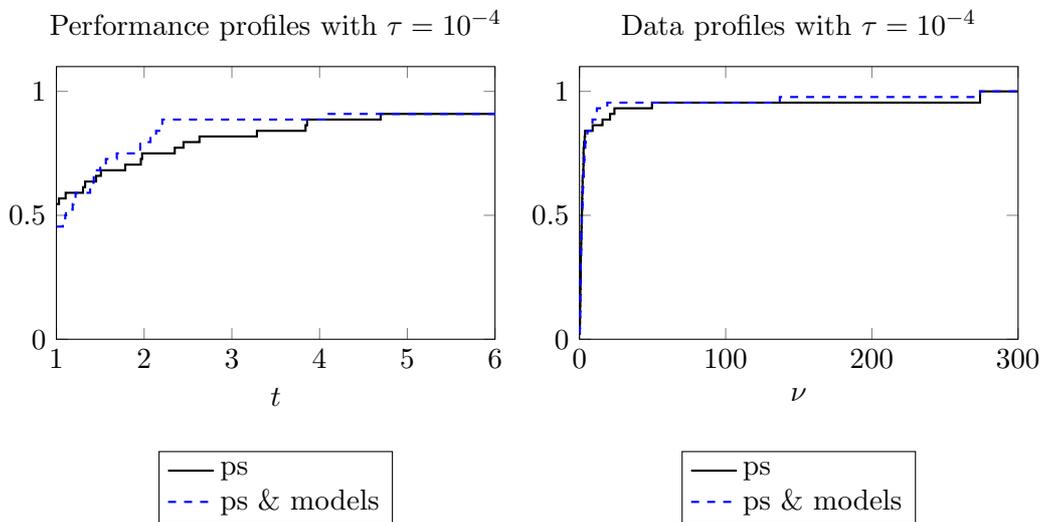


Figure 4.6: Performance and data profiles for structured variants, with or without models (smallish test set).

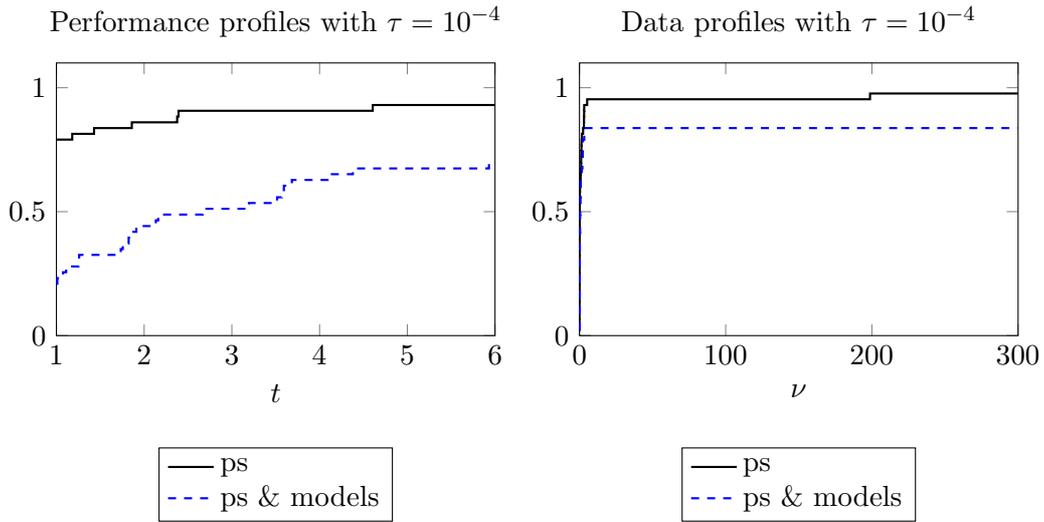


Figure 4.7: Performance and data profiles for structured variants, with or without models (medium test set).

less efficient in function evaluations). Because of the need to store the model itself, memory usage can significantly increase for the variant using unstructured models (at least for fully quadratic ones as discussed above). This is also the case for the structured variants because they have to store, analyze (once) and exploit the structure, which requires using additional pointers and lists. However, the additional memory necessary to use models in the structured case remains typically modest, as only a collection of small matrices needs being stored. We finally note that variants using models require a significantly higher internal computational effort, mostly in the solution of the many trust-region subproblems involved in managing the interpolation set and computing the search step. This is apparent in Figure 4.7 and in Table 1.3 in Appendix A where we report the number of function evaluations taken by all variants on all test sets using the internal default stopping criterion. It is clear from the table that the worse reliability of the 'ps & models' variant is nearly entirely due to exhausting the maximum allowed cpu-time (symbol '†' in the table). Numerical experiments were also carried out using linear and diagonal models in the search step (both in the structured and unstructured cases). Unfortunately, it was observed that, despite the solution of the involved trust-region problem being considerably cheaper than when using quadratic models, the use of the simpler models resulted in a considerable increase of the number of function evaluations, and thus worse overall performance.

## 5 Conclusions and perspectives

We have introduced a structured version of derivative-free random pattern search algorithms which is able to exploit coordinate partially separable structure (typically as-

sociated with sparsity) present in unconstrained and bound-constrained optimization problems. This techniques improves performance by orders of magnitude and makes it possible to solve problems that otherwise are totally intractable by standard derivative-free methods.

We have also described a library of interpolation-based modelling tools which can be associated to the structured or unstructured versions of the initial pattern search algorithm. For problems of small or moderate size, the use of the library further enhances performance, especially when associated with structure. For larger problems the internal computing costs increase and, while still reducing the number of function evaluations, the use of the library may require a sometimes unrealistic computing time, in particular if the problem is unstructured.

In comparing the benefits of using problem structure in a poll step and building local models using interpolation techniques, we have concluded that the former is likely to be more efficient, in particular for larger problems, even for structure-exploiting models.

A new release of the Matlab BFO package<sup>(11)</sup> featuring both use of structure and modelling tools (as discussed in this paper) is now freely available online from

<https://github.com/m01marpor/BFO>

and the main new features are briefly sketched in Appendix D.

The selective use of structured models in conjunction with structured poll steps remains an attractive option. However, the criteria defining the circumstances in which interpolation models should be used, if at all, need further investigation. Many other topics also merit research, including the design of “grayer” optimization tools which could exploit derivatives available for part of the problem while adapting the techniques described here for the rest.

## Acknowledgements

Both authors are indebted to the University of Florence for its support while the present research was being conducted.

The first author is member of the *Gruppo Nazionale per il Calcolo Scientifico* (GNCS) of the Istituto Nazionale di Alta Matematica (INdAM) and this work was partially supported by INdAM-GNCS under Progetti di Ricerca 2019 and 2020.

## References

- [1] M. A. Abramson, C. Audet, J. W. Chrissis, and J. G. Walston. Mesh adaptive direct search algorithms for mixed variable optimization. *Optimization Letters*, 3(1):35–47, 2009.
- [2] C. Audet and J. E. Dennis. Pattern search algorithms for mixed variable programming. *SIAM Journal on Optimization*, 11(3):573–594, 2001.

---

<sup>(11)</sup>Although this is unrelated to the subject of the present paper, we emphasize that the new BFO release also features the important capability of handling categorical variables, as well as the new training strategies discussed in [43], see Appendix D.

- [3] C. Audet and W. Hare. *Derivative-free and blackbox optimization*. Springer Verlag, Heidelberg, Berlin, New York, 2017.
- [4] C. Audet, S. Le Digabel, and Ch. Tribes. The mesh adaptive direct search algorithm for granular and discrete variables. *SIAM Journal on Optimization*, 29(2):1164–1189, 2019.
- [5] B. M. Averick and J. J. Moré. The Minpack-2 test problem collection. Technical Report ANL/MCS-P153-0694, Mathematics and Computer Science, Argonne National Laboratory, Argonne, Illinois, USA, 1992.
- [6] G. E. P. Box and K. B. Wilson. On the experimental attainment of optimum conditions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 13(1):1–45, 1951.
- [7] C. Cartis, J. Fiala, B. Marteau, and L. Roberts. Improving the flexibility and robustness of model-based derivative-free optimization solvers. *ACM Transactions on Mathematical Software*, 45(3):32, 2019.
- [8] C. Cartis, N. I. M. Gould, and Ph. L. Toint. Trust-region and other regularization of linear least-squares problems. *BIT*, 49(1):21–53, 2009.
- [9] B. Colson and Ph. L. Toint. Exploiting band structure in unconstrained optimization without derivatives. *Optimization and Engineering*, 2:349–412, 2001.
- [10] B. Colson and Ph. L. Toint. A derivative-free algorithm for sparse unconstrained optimization problems. In A. H. Siddiqi and M. Kočvara, editors, *Trends in Industrial and Applied Mathematics*, pages 131–149, Dordrecht, The Netherlands, 2002. Kluwer Academic Publishers.
- [11] B. Colson and Ph. L. Toint. Optimizing partially separable functions without derivatives. *Optimization Methods and Software*, 20(4-5):493–508, 2005.
- [12] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-Region Methods*. MPS-SIAM Series on Optimization. SIAM, Philadelphia, USA, 2000.
- [13] A. R. Conn, K. Scheinberg, and Ph. L. Toint. On the convergence of derivative-free methods for unconstrained optimization. In A. Iserles and M. Buhmann, editors, *Approximation Theory and Optimization: Tributes to M. J. D. Powell*, pages 83–108, Cambridge, England, 1997. Cambridge University Press.
- [14] A. R. Conn, K. Scheinberg, and L. N. Vicente. Geometry of sample sets in derivative free optimization. Part I: polynomial interpolation. Technical Report 03-09, Departamento de Matemática, Universidade de Coimbra, Portugal, 2003. Revised September 2004.
- [15] A. R. Conn, K. Scheinberg, and L. N. Vicente. Geometry of interpolation sets in derivative free optimization. *Mathematical Programming, Series B*, 111(1-2):141–172, 2008.
- [16] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to Derivative-free Optimization*. MPS-SIAM Series on Optimization. SIAM, Philadelphia, USA, 2009.
- [17] A. R. Conn and Ph. L. Toint. An algorithm using quadratic interpolation for unconstrained derivative free optimization. In G. Di Pillo and F. Gianessi, editors, *Nonlinear Optimization and Applications*, pages 27–47, New York, 1996. Plenum Publishing.
- [18] I. D. Coope and C. J. Price. Frame-based methods for unconstrained optimization. *Journal of Optimization Theory and Applications*, 107(2):261–274, 2000.
- [19] I. D. Coope and C. J. Price. On the convergence of grid-based methods for unconstrained optimization. *SIAM Journal on Optimization*, 11(4):859–869, 2001.
- [20] I. D. Coope and C. J. Price. Positive basis in numerical optimization. *Computational Optimization and Applications*, 21(2):169–175, 2002.
- [21] A. L. Custódio, H. Rocha, and L. N. Vicente. Incorporating minimum Frobenius norm models in direct search. *Computational Optimization and Applications*, 46(2):265–278, 2010.
- [22] J. E. Dennis and V. Torczon. Direct search methods on parallel machines. *SIAM Journal on Optimization*, 1(4):448–474, 1991.

- [23] E. D. Dolan, R. M. Lewis, and V. Torczon. On the local convergence of pattern search. *SIAM Journal on Optimization*, 14(2):567–583, 2003.
- [24] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.
- [25] E. D. Dolan, J. J. Moré, and T. S. Munson. Optimality measures for performance profiles. *SIAM Journal on Optimization*, 16(3):891–909, 2006.
- [26] 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.
- [27] S. Gratton, C. W. Royer, L. N. Vicente, and Z. Zhang. Direct search based on probabilistic descent. *SIAM Journal on Optimization*, 25(3):1515–1541, 2015.
- [28] S. Gratton, Ph. L. Toint, and A. Tröltzsch. An active-set trust-region method for derivative-free nonlinear bound-constrained optimization. *Optimization Methods and Software*, 21(4-5):873–894, 2011.
- [29] L. Grippo and F. Rinaldi. A class of derivative-free nonmonotone optimization algorithms employing coordinate rotations and gradient approximations. *Computational Optimization and Applications*, 60(1):1–33, 2014.
- [30] R. Hooke and T. A. Jeeves. Direct search solution of numerical and statistical problems. *Journal of the ACM*, 8:212–229, 1961.
- [31] J. C. Lagarias, B. Poonen, and M. H. Wright. Convergence of the restricted Nelder–Mead algorithm in two dimensions. *SIAM Journal on Optimization*, 22:501–532, 2012.
- [32] J. Larson, M. Menickelly, and S. M. Wild. Derivative-free optimization methods. *Acta Numerica*, 28:287–404, 2019.
- [33] S. Le Digabel. Algorithm 909: NOMAD: Nonlinear optimization with the MADS algorithm. *ACM Transactions on Mathematical Software*, 37(4):1–44, 2011.
- [34] R. M. Lewis and V. Torczon. Pattern search methods for linearly constrained minimization. *SIAM Journal on Optimization*, 10(3):917–941, 2000.
- [35] G. Liuzzi, S. Lucidi, and F. Rinaldi. An algorithmic framework based on primitive directions and non-monotone line searches for black box problems with integer variables. *Mathematical Programming Computation*, 12:673–702, 2020.
- [36] S. Lucidi and M. Sciandrone. On the global convergence of derivative-free methods for unconstrained optimization. *SIAM Journal on Optimization*, 13(1):97–116, 2002.
- [37] L. Marini, B. Morini, and M. Porcelli. Quasi-Newton methods for constrained nonlinear systems: complexity analysis and applications. *Computational Optimization and Applications*, 71(1):147–170, 2018.
- [38] J. J. Moré and D. C. Sorensen. Computing a trust region step. *SIAM Journal on Scientific and Statistical Computing*, 4(3):553–572, 1983.
- [39] J. J. Moré and S. M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization*, 20(1):172–191, 2009.
- [40] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.
- [41] A. Neumaier, H. Fendl, H. Schilly, and T. Leitner. VXQR: Derivative-free unconstrained optimization based on QR factorizations. *Soft Computing*, 15(11): 2287–2298, 2011.
- [42] 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.

- [43] 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.
- [44] M. J. D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In S. Gomez and J. P. Hennart, editors, *Advances in Optimization and Numerical Analysis, Proceedings of the Sixth Workshop on Optimization and Numerical Analysis, Oaxaca, Mexico*, volume 275, pages 51–67, Dordrecht, The Netherlands, 1994. Kluwer Academic Publishers.
- [45] M. J. D. Powell. A direct search optimization method that models the objective by quadratic interpolation. Presentation at the 5th Stockholm Optimization Days, Stockholm, 1994.
- [46] M. J. D. Powell. Trust region methods that employ quadratic interpolation to the objective function. Presentation at the 5th SIAM Conference on Optimization, Victoria, 1996.
- [47] M. J. D. Powell. Direct search algorithms for optimization calculations. *Acta Numerica*, 7:287–336, 1998.
- [48] M. J. D. Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. Technical report, Department of Applied Mathematics and Theoretical Physics, Cambridge University, Cambridge, England, 2009.
- [49] C. J. Price and Ph. L. Toint. Exploiting problem structure in pattern-search methods for unconstrained optimization. *Optimization Methods and Software*, 21(2):479–491, 2006.
- [50] P. Richtárik and M. Takáč. Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, 156(1-2):433–484, 2016.
- [51] L. Roberts and C. Cartis. A derivative-free gauss–newton method. *Mathematical Programming Computation*, 11(4):631–674, 2019.
- [52] H. H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *Computer Journal*, 3:175–184, 1960.
- [53] Ph. R. Sampaio and Ph. L. Toint. A derivative-free trust-funnel method for equality-constrained nonlinear optimization. *Computational Optimization and Applications*, 61(1):25–49, 2015.
- [54] Ph. R. Sampaio and Ph. L. Toint. Numerical experience with a derivative-free trust-funnel method for nonlinear optimization problems with general nonlinear constraints. *Optimization Methods and Software*, 31(3):511–534, 2016.
- [55] K. Scheinberg and Ph. L. Toint. Self-correcting geometry in model-based algorithms for derivative-free unconstrained optimization. *SIAM Journal on Optimization*, 20(6):3512–3532, 2010.
- [56] T. Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20(3):626–637, 1983.
- [57] Ph. L. Toint. Towards an efficient sparsity exploiting Newton method for minimization. In I. S. Duff, editor, *Sparse Matrices and Their Uses*, pages 57–88, London, 1981. Academic Press.
- [58] V. Torczon. On the convergence of the multidirectional search algorithm. *SIAM Journal on Optimization*, 1(1):123–145, 1991.
- [59] V. Torczon. On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, 7(1):1–25, 1997.
- [60] D. Winfield. Function minimization by interpolation in a data table. *Journal of the Institute of Mathematics and its Applications*, 12:339–347, 1973.
- [61] S. J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

## A Full results

Table 1.3 shows the number of objective function evaluations required by the structured/unstructured versions of BFO with and without interpolation model for the problem instances of Table 4.1 ( $\infty$  meaning that more than 100000 evaluations were needed,  $\dagger$  meaning that cpu time exceed the maximum time, 8 hours, allowed). Moreover, a blank entry means that the solution of the corresponding problem instance was not attempted either because the solver version failed to solve a smaller version of the same problem (symbols  $\dagger$  and  $\infty$ ) or because the problem size was considered too large for the considered solver. In particular, the unstructured/model implementation was used for solving small problems only and the partially separable/model implementation was not applied to solve large problems).

## B Details for problem CONTACT

The problem CONTACT is a bilinear finite-element discretization of a membrane contact problem defined on the unit square of  $\mathbb{R}^2$ . This square is discretized in  $q = (\sqrt{n} - 1)^2$  element squares and the surface of the membrane “above” this element square is given by

$$\frac{1}{q} \sqrt{1 + (x_{SW} - x_{NE})^2 + (x_{SO} - x_{NW})^2},$$

where  $x_{SW}$ ,  $x_{NE}$ ,  $x_{SO}$  and  $x_{NW}$  give the height of the membrane at the four corners of the element square. The membrane is fixed on the boundaries of the unit square to the value of the function

$$b(x, y) = 1 + 8x + 4y + 3 \sin(2\pi x) \sin(2\pi y). \quad (2.1)$$

In addition, the membrane has to lie above a square flat obstacle of height 10 positioned at  $[0.4, 0.6]^2$ . The starting point is the projection of the vector given by the values of (2.1) at each element square’s corner onto the feasible domain.

## C Details for problem NZF1

The NZF1 problem is a variable dimension version of the eponymous nonlinear least-squares problem defined in [49] as an example illustrating the concept of partially sepa-

Pb	$n$	unstructured		partially separable		Pb	$n$	unstructured		partially separable	
		no mod	model	no mod	model			no mod	model	no mod	model
ARWHEAD	10	962	365	79	148	BDARWHD	10	1199	1227	101	153
	50	11859		91	122		50	$\infty$		70	381
	100	36085		97	214		100			71	345
	500	$\infty$		146	659		500			81	691
	1000			194	1015		1000			76	869
	5000			389			5000			77	
	10000			618			10000			77	
BDEXP	10	21122	222	1661	375	BDQRTIC	10	2468	883	290	345
	50	$\infty$		13218	828		50	$\infty$		301	1189
	100			34408	156		100			298	457
	500			$\infty$	137		500			393	373
	1000				132		1000			542	503
	5000						5000			1150	
	10000						10000			1480	
BEALES	10	$\infty$	$\infty$	275	70	BROYDN3D	10	1225	456	308	96
	50			275	85		50	22244		199	118
	100			275	123		100	74758		273	151
	500			275	203		500	$\infty$		304	215
	1000			275	234		1000			370	303
	5000			300			5000			640	
	10000			325			10000			675	
BROWNAL6	10	6171	1224	15773	567	CONTACT	16	268	126	265	107
	50	$\infty$		4325	816		64	30452		221	248
	100			2788	1798		144	$\infty$		442	494
	500			2682	†		400			1009	1295
	1000			2847	†		900			1761	†
	5000			3059			2500			3589	
	10000			3118			4900	$\infty$		5952	
DIXMAANA	15	2358	558	185	113	DIXMAANI	15	17702	1405	185	152
	51	17357		189	101		51	$\infty$		186	195
	102	55748		240	96		102			184	284
	501	$\infty$		226	313		501			230	1231
	1002			375	869		1002			265	1487
	5001			310			5001			709	
	10002			756			10002			798	
ENGVAL	10	1567	629	143	83	FREUROTH	10	83101	4570	257	2297
	50	30400		155	94		50	$\infty$		181	2183
	100	$\infty$		157	129		100			191	2712
	500			159	158		500			185	†
	1000			159	218		1000			192	†
	5000			159			5000			233	
	10000			159			10000			317	
HELIX	11	9931	2094	131	101	JNLBRNG1	24	734	65	101	19
	51	$\infty$		152	113		84	27567		301	779
	101			167	108		180	$\infty$		427	37
	501			298	122		544			944	86
	1001			388	152		1012			1393	92
	5001			883			1984			1909	
	10001			1070			5304			1799	
LMINSURF	16	318	117	306	76	MOREBV	12	7154	391	7010	117
	64	23692		461	243		52	22732		60	242
	144	$\infty$		1065	516		102	1923		47	263
	400			3301	1580		502	9001		47	55
	900			8506	†		1002	18001		47	56
	2500			29961			5002	90001		47	
	4900			$\infty$			10002	$\infty$		47	
NLMINSRF	16	415	94	412	76	NZF1	13	1480	1397	177	297
	64	29409		579	358		39	61772		225	347
	144	$\infty$		1082	623		130	$\infty$		625	523
	400			8853	1824		650			684	1476
	900			3412	†		1300			667	2185
	2500			31619			6500			946	
	4900			$\infty$			13000			2228	
POWSING	20	20605	2110	716	776	ROSENBR	10	13241	12229	361	299
	52	$\infty$		824	81		50	$\infty$		361	534
	100			849	84		100			384	729
	500			988	94		500			436	1531
	1000			1036	98		1000			461	2334
	5000			1164			5000			636	
	10000			1148			10000			736	
TRIDIA	10	3073	139	440	17	WOODS	20	$\infty$	32414	1609	1538
	50	$\infty$		316	20		40			1747	1985
	100			314	23		200			1924	4210
	500			345	30		500			2241	†
	1000			293	33		1000			2927	†
	5000			277			5000			4515	
	10000			278			10000			5002	

Table 1.3: The number of objective function evaluations required by the structured/unstructured versions of BFO with and without interpolation model for the problem instances of Table 4.1.

rable functions. If  $n = 13\ell$ , it is defined by

$$\begin{aligned}
 f(x_1, \dots, x_n) = & \sum_{i=1}^{\ell} \left[ \left( 3x_i - 60 + \frac{1}{10}(x_{i+1} - x_{i+2})^2 \right)^2 \right. \\
 & + \left( x_{i+1}^2 + x_{i+2}^2 + x_{i+3}^2(1 + x_{i+3})^2 + x_{i+6} + \frac{x_{i+5}}{(1 + x_{i+4}^2 + \sin(\frac{x_{i+4}}{1000}))} \right)^2 \\
 & + \left( x_{i+6} + x_{i+7} - x_{i+8}^2 + x_{i+10} \right)^2 \\
 & + \left( \log(1 + x_{i+10}^2) + x_{i+11} - 5x_{i+12} + 20 \right)^2 \\
 & \left. + \left( x_{i+4} + x_{i+5} + x_{i+5} * x_{i+9} + 10x_{i+9} - 50 \right)^2 \right] \\
 & + \sum_{i=1}^{\ell-1} (x_{i+6} - x_{i+19})^2.
 \end{aligned}$$

The starting point is the vector of all ones.

## D BFO 2.0 and its new features

Release 2.0 of the Matlab BFO package is a major upgrade from Release 1 and includes several important new problem-oriented possibilities. Beyond the BFOSS library of model-based search steps described in Section 3 and the exploitation of CPS structure described in Section 2, BFO 2.0 also supports the following new problem features.

**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

$$\{\{ \text{'blue'}, 3.1416, \text{'magenta'}, 2 \}\}.$$

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 neighbourhood 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 [42] for details). Release 2.0 now includes two new training strategies (introduced in [43]):

**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 [24, 25].

**Data profiling.** This option is similar to performance profiling, but uses data profiles [39] 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.