A Model-Based Derivative-Free Optimization Algorithm for Partially Separable Problems

YICHUAN LIU, School of Mathematical Sciences, Fudan University, China

YINGZHOU LI, School of Mathematical Sciences, Fudan University, China, Shanghai Key Laboratory for Contemporary Applied Mathematics, China, and Key Laboratory of Computational Physical Sciences (MOE), China

We propose UPOQA, a derivative-free optimization algorithm for partially separable unconstrained problems, leveraging quadratic interpolation and a structured trust-region framework. By decomposing the objective into element functions, UPOQA constructs underdetermined element models and solves subproblems efficiently via a modified projected gradient method. Innovations include an approximate projection operator for structured trust regions, improved management of elemental radii and models, a starting point search mechanism, and support for hybrid black-white-box optimization, etc. Numerical experiments on 85 CUTEst problems demonstrate that UPOQA can significantly reduce the number of function evaluations. To quantify the impact of exploiting partial separability, we introduce the speed-up profile to further evaluate the acceleration effect. Results show that the acceleration benefits of UPOQA are marginal in low-precision scenarios but become more pronounced in high-precision scenarios. Applications to quantum variational problems further validate its practical utility.

CCS Concepts: • **Mathematics of computing → Solvers**; **Nonconvex optimization**.

Additional Key Words and Phrases: Partially separable problems, derivative-free optimization, trust region methods, interpolation model

ACM Reference Format:

Yichuan Liu and Yingzhou Li. 2025. A Model-Based Derivative-Free Optimization Algorithm for Partially Separable Problems. *J. ACM* 37, 4, Article 111 (August 2025), 27 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Derivative-free optimization (DFO) [15] methods, also known as black-box optimization (BBO) or zero-order optimization methods, are designed to find the minimum or maximum of an objective function when its derivatives are not available. For complex problems, the objective function may not be expressed as a closed-form analytical expression but rather as a black box that only allows function evaluations. Examples include cases where function values are obtained from simulation or experimental observations, or from the output of software packages with unknown internal implementations. Such objective functions often incur high evaluation costs, yield noisy values, or both.

DFO algorithms can be broadly categorized into five classes. The first class comprises *direct search* and *pattern search* methods [35, 36], which select the next iteration point by comparing the relative merits of function values [27]. These methods typically employ specific geometric patterns for point selection. They impose minimal requirements

Authors' addresses: Yichuan Liu, liuyichuan2020@outlook.com, School of Mathematical Sciences, Fudan University, Shanghai, China, 200433; Yingzhou Li, yingzhouli@fudan.edu.cn, School of Mathematical Sciences, Fudan University, Shanghai, China, 200433 and Shanghai Key Laboratory for Contemporary Applied Mathematics, Shanghai, China, 200433 and Key Laboratory of Computational Physical Sciences (MOE), Shanghai, China, 200433.

Manuscript submitted to ACM

on the properties of the objective function, are computationally simple, and less prone to local minima, but often exhibit slow convergence. The second class consists of *line-search-based* methods, with Powell's method [41] being a representative example that performs line searches along conjugate direction sets. The third class includes *heuristic search* methods, which explore the solution space following specific heuristic rules to find approximate optima. Common algorithms in this category are simulated annealing [32] and evolutionary algorithms [17, 54], with `CMA-ES` being a notable example [26]. The fourth class is *model-based derivative-free optimization* [7, 16, 29, 51]. These methods construct a surrogate model through interpolation or regression using known function values, approximating the objective function locally. The model is then optimized and updated within a trust-region framework to solve the original problem. Examples include Powell's `COBYLA` [43] (based on first-order models) and `BOBYQA` [48], `NEWUOA` [46] (based on second-order models), etc. Surrogate models may also adopt other forms, such as radial basis functions [39] and moving ridge functions (e.g., `OMoRF` [25]). The fifth class comprises *finite-difference-based* methods[1], which approximate the true gradient using numerical differentiation and then employ gradient-based optimization algorithms.

In DFO, the primary challenge lies in the lack of structural knowledge about the problem. When the problem lacks smoothness, convexity, or is subject to significant noise, many DFO methods essentially sample the parameter space according to certain patterns. Such approaches are prone to the curse of dimensionality when dealing with large-scale problems. Even if smoothness is assumed and surrogate models are constructed to accelerate convergence, the computational cost of solving these problems may still be prohibitive [53]. To address this, many researchers have considered structured DFO problems, such as nonlinear least squares [7, 28, 42, 58], problems with sparse Hessian matrices [9, 10, 53] and partially separable problems [5, 11, 40, 49, 55, 60]. Exploiting these structures can significantly reduce the number of function evaluations required for convergence.

This paper focuses particularly on unconstrained partially separable problems [22], which take the form:

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

where $f_1, \ldots, f_q$ are referred to as *element functions* (or simply *elements*). For each $i = 1, \ldots, q$, there exists a subspace $\mathcal{N}_i \subset \mathbb{R}^n$ such that for any $w \in \mathcal{N}_i$ and $x \in \mathbb{R}^n$, the following holds:

$$f_i(x + w) = f_i(x).$$

In contrast to the elements, f is called the *overall function*. Let $\mathcal{R}_i \triangleq \mathcal{N}_i^{\perp}$, and denote $n_i \triangleq \dim \mathcal{R}_i$ as the *elemental dimension* of f_i. Each f_i is essentially a function defined on $\mathcal{R}_i$. The corresponding projection is denoted as $P_{\mathcal{R}_i} : \mathbb{R}^n \to \mathcal{R}_i$, ensuring that for any $x \in \mathbb{R}^n$, $f_i(x) \equiv f_i(P_{\mathcal{R}_i}(x))$. Without loss of generality, we assume $\operatorname{span}\left(\cup_{i=1}^{q} \mathcal{R}_i\right) = \mathbb{R}^n$, as otherwise, redundant variables would not affect the value of f.

Partially separable structures frequently arise in optimal control problems, the discretization of partial differential equations (PDEs) via methods like finite elements, and of other variational problems [23]. In optimal control, problems often consist of multiple loosely coupled subsystems distributed across space and time. Similarly, problems stemming from domain decomposition techniques applied to PDE discretizations exhibit partial separability, where each variable primarily interacts with its neighboring regions. In quantum computing, many variational quantum problems also demonstrate such structures, with each subfunction represented by a quantum circuit [3, 4].

In practice, the more commonly used concept is *coordinate partial separability*, which further requires that each f_i depends only on a subset of variables $\left\{x_j \mid j \in \mathcal{I}_i = \{j_1, \ldots, j_{n_i}\}\right\}$, referred to as the *elemental variables* of f_i. In this case, $\mathcal{R}_i = \operatorname{span}(\{e_j \mid j \in \mathcal{I}_i\})$, and we denote $P_{\mathcal{R}_i}(x)$ as $x^{\mathcal{I}_i}$. Since $f_i(x)$ depends solely on $x^{\mathcal{I}_i}$, we will, for simplicity,

no longer distinguish between the notations $f_i(x^{\mathcal{I}_i})$ and $f_i(x)$ in subsequent discussions. Due to its prevalence and algorithmic convenience, unless explicitly stated otherwise, all references to partial separability in this paper refer to coordinate partial separability. We further assume that each element function can be evaluated independently. This assumption aligns with the practical context of most problems, as when each f_i represents the contribution of a subsystem, these contributions should naturally be computable separately.

The term *partial* in partial separability refers to the fact that the variables on which different element functions depend may overlap. The two extremes of this property are when all elements depend on completely different variables, such as $f(x) = \sum_{i=1}^{q} f_i(x_i)$, and when all elements depend on all variables, i.e., $f(x) = \sum_{i=1}^{q} f_i(x)$. However, most real-world problems lie between these two cases [19, 37, 56]. In the `LANCELOT` package [14], the concept of partial separability is further generalized to *group partial separability* and used to define the *standard input format* (SIF) for test problems. Moreover, many common problem structures can be viewed as special cases of partially separable structures. Any twice continuously differentiable function with a sparse Hessian can be transformed into a partially separable form [22], and nonlinear least squares problems $\min_x \sum_{i=1}^{N} r_i(x)^2$ are naturally partially separable.

The most obvious advantage of partial separability is that, given a trial point $x \in \mathbb{R}^n$, one can always compute the value of any element function $f_i(x)$ individually without evaluating the full $f(x)$. Conversely, even if $f(x)$ must be computed, all element values $f_1(x), \ldots, f_q(x)$ can be obtained simultaneously, thereby significantly increasing the amount of information available. Various methods have been developed to exploit partial separability. When derivatives are available, a common approach is to employ partitioned Hessian updating schemes [6, 23, 24, 33, 57] in quasi-Newton methods, along with techniques such as element merging and grouping [13, 30], element-wise preconditioning [18], and parallelization [33] to reduce the computational cost of Newton steps or conjugate gradients. Conn et al. proposed a trust-region framework [12] for solving partially separable problems, which was later extended by Shahabuddin [55] with three algorithm variants based on different methods for solving the trust-region subproblems. Another approach is *incremental optimization* [2, 30], where only a small subset of elements is optimized in each iteration.

When derivatives are unavailable, existing approaches can be broadly categorized into *bottom-up* and *top-down* approaches, depending on whether they directly optimize the element functions or optimize the objective function. The *bottom-up* approach primarily leverages the independent evaluability of elements to reconstruct information about the objective function from element values at relatively low cost, thereby solving problem (1). A straightforward implementation evaluates elements at grid points in $\mathbb{R}^n$, e.g., $\{x = \sum_{i=1}^{n} c_i e_i \mid c_i \in \mathbb{Z}\}$, then assembles these evaluations into full function values. For instance, for $f(x, y, z) = f_1(x, y) + f_2(y, z)$, computing the values of f_i at $\{0, 1\}^2$ requires only 4 evaluations but yields all 8 values of f over $\{0, 1\}^3$. If elements are independent, evaluating each element k times produces k^q objective values from just kq evaluations. Price and Toint [49] exploited this to develop a mesh-refinement-based direct search method that scales efficiently to thousands of dimensions. Another strategy is line-search-based, such as Porcelli and Toint's `BFO` method [40].

An alternative approach is *top-down*, where the algorithm primarily extracts information from complete evaluations of f to guide the optimization. Model-based methods often fall into this category, as the subproblems formed by surrogate models typically generate points without a clear geometric pattern, making it unlikely to reconstruct the objective value by computing only a few element values. Relevant work includes Bouzarkouna et al.'s `p-sep lmm-CMA` method [5] and Conn and Toint's `PSDFO` method [11]. The `p-sep lmm-CMA` method is a variant of `lmm-CMA` [31], where a meta-model (a second-order polynomial regression model) is built for each element function to guide the ranking of data points in `lmm-CMA`. Meanwhile, `PSDFO` is an interpolation-based trust-region method where each element f_i is approximated by a quadratic interpolation model m_i. In both methods, the primary purpose of modeling elements is to

construct an *overall model*

$$m(x) = \sum_{i=1}^{q} m_i(x), \tag{2}$$

which is then used as a single surrogate model in the underlying algorithm for general optimization. Consequently, the information available to the algorithm mainly stems from feedback on the full function f or the overall model m. The benefits of the *top-down* approach are relatively indirect. Compared to using a single surrogate model, employing multiple element models primarily reduces the total modeling cost, since evaluating f once yields all values $f_1, \ldots, f_q$, enabling simultaneous updates to all models in a single iteration.

Generally speaking, model-based methods exhibit superior computational efficiency compared to direct search and pattern search approaches. In pursuit of efficiency, we adopt a *top-down* strategy and develop a trust-region algorithm that employs quadratic interpolation models to approximate element functions. The algorithm is named *Unconstrained Partially-separable Optimization by Quadratic Approximation* (UPOQA). The key features of UPOQA include:

(1) For interpolation modeling, UPOQA utilizes the underdetermined quadratic model based on Powell's *derivative-free symmetric Broyden update* [44], incorporating Powell's techniques [45, 47] to enable efficient model construction and maintenance with low algebraic complexity;
(2) For trust-region management, UPOQA assigns independent trust-region radii to each element, implements a modified projected gradient method for solving trust-region subproblems within structured trust regions, improves upon the existing radius adjustment strategy [55], and introduces a selective update mechanism for element models based on obtained interpolation points;
(3) In terms of accessibility, UPOQA provides a ready-to-use Python implementation with various features to enhance flexibility and robustness, including hybrid black-white-box optimization and restart mechanism. The open-source package is released on GitHub under the BSD 3-Clause License.

The rest of the paper is organized as follows. Section 2 presents the overall framework of UPOQA, including the designed approximate projection operator for structured trust-region problems, the management of trust-region radius, the criteria for selective element model updates, starting point search mechanism, and optional features (restart and hybrid optimization). Section 3 conducts comprehensive numerical experiments on CUTEst test problems extracted via the S2MPJ tool [21] and test cases from the quantum variational problem (17) to validate the algorithm's effectiveness. Section 4 concludes the work and discusses potential future research directions.

2 THE UPOQA ALGORITHM

The UPOQA algorithm is based on Powell's trust-region framework [46, 48, 51] for solving problem (1). For each element f_i, the algorithm maintains an interpolation set $\mathcal{Y}_i$ of size at least $O(n_i)$, along with an underdetermined quadratic interpolation model [44] constructed from these points:

$$m_{k,i}(x_k^{\mathcal{I}_i} + s) = c_{k,i} + g_{k,i}^\top s + \frac{1}{2} s^\top H_{k,i} s, \quad s \in \mathbb{R}^{n_i}. \tag{3}$$

We refer to $m_{k,1}, \ldots, m_{k,q}$ as *element models*, where $c_{k,i} \in \mathbb{R}$, $g_{k,i} \in \mathbb{R}^{n_i}$, and $H_{k,i} \in \mathbb{R}^{n_i \times n_i}$ is a symmetric matrix. The model $m_{k,i}$ is expected to sufficiently approximate the objective function within a trust region of radius $\Delta_{k,i}$:

$$\mathcal{B}_i(\Delta_{k,i}) = \left\{ x_k^{\mathcal{I}_i} + s \mid s \in \mathbb{R}^{n_i},\ \|s\|_2 \le \Delta_{k,i} \right\}.$$

Moreover, each model is equipped with its own trust-region radius $\{\Delta_{k,i}\}_{i=1}^{q}$ to account for their varying reliability due to different accuracies.

During the execution of UPOQA, as the interpolation sets are updated, the element models are also updated by the *derivative-free symmetric Broyden update* [44], which has been widely employed by Powell in algorithms such as NEWUOA [46] and BOBYQA [48]. It allows the algorithm to use only $O(n_i)$ interpolation points per model while updating the model with $O(n_i^2)$ algebraic complexity [45, 47]. This surrogate model is widely recognized as an effective second-order approach in derivative-free methods based on interpolation models and trust regions [15].

Our work is tailored for partially separable problems. When the number of elements exceeds one, two key issues must be addressed. First, how to solve the resulting structured trust-region subproblem:

$$\begin{aligned} \min_{s\in\mathbb{R}^n} \quad & m_k(x_k+s), \\ \text{s.t.} \quad & \|s^{\mathcal{I}_i}\|_2 \le \Delta_{k,i}, \quad i=1,2,\ldots,q, \end{aligned} \tag{4}$$

where $m_k \triangleq \sum_{i=1}^{q} m_{k,i}$ is the *overall model* assembled from the element models $\{m_{k,i}\}_{i=1}^{q}$. Second, how to evaluate the contribution of each element model to the reduction of the objective value in each iteration and accordingly adjust $\{\Delta_{k,i}\}_{i=1}^{q}$. For the first issue, we employ a modified projected gradient method based on an approximate projection operator, which will be detailed in Section 2.1. For the second issue, we directly adopt the combined separation criterion proposed by Shahabuddin [55, Section 2.4], with minor modifications in corner cases.

Next, we briefly outline the general workflow of the UPOQA algorithm for solving problem (1). At initialization, the algorithm first constructs the surrogate model $m_{0,i}$ and the interpolation set $\mathcal{Y}_{0,i}$ for each element f_i. Then, using all interpolation points from $\{\mathcal{Y}_{0,i}\}_{i=1}^{q}$, it seeks a better starting point x_0 via the method described in Section 2.4. If successful, the objective value $f(x_0)$ will be lower than that of the user-provided point x_{start}. In each iteration, UPOQA first solves the trust-region subproblem (4) using the modified projected gradient method from Section 2.1, obtaining a trial step s_k. The algorithm then decides whether to accept this step based on the magnitude of $\|s_k\|_2$.

For s_k with large norms, the algorithm first computes a candidate point $\hat{x}_k = x_k + s_k$ and updates all models and interpolation point sets based on $\hat{x}_k$. When updating the model $m_{k,i}$ and the interpolation set $\mathcal{Y}_{k,i}$, a new interpolation point $\hat{x}_k^{\mathcal{I}_i}$ is added to $\mathcal{Y}_{k,i}$. However, this procedure is performed only if $\hat{x}_k^{\mathcal{I}_i}$ does not compromise the poisedness of the interpolation set $\mathcal{Y}_{k,i}$. For updating the trust-region radii $\{\Delta_{k,i}\}_{i=1}^{q}$, we employ the combined separation criterion proposed by Shahabuddin [55, Section 2.4]. If the objective exhibits insufficient reduction at $\hat{x}_k$, the algorithm further checks whether a geometry-improving step should be taken. If the conditions for a geometry-improving step are not met, the trust-region resolution ρ_k is reduced. The value of ρ_k serves as a lower bound for all trust-region radii $\{\Delta_{k,i}\}_{i=1}^{q}$, indicating the search precision of the algorithm at the current iteration. For s_k with small norms, the algorithm will reduce the trust-region radii and perform geometry-improvement steps. If such small-step events occur consecutively multiple times or no geometry improvement steps are triggered, the algorithm also reduces ρ_k.

Algorithm 1 outlines the procedure of UPOQA. Steps unrelated to the partially separable structure closely resemble those in Powell-style model-based algorithms. For these aspects, we refer readers to the descriptions of these algorithms [46, 47, 50, 51] or the implementation of UPOQA for further details.

2.1 Structured Trust Region

UPOQA assigns an independent trust-region radius $\Delta_{k,i}$ to each element model $m_{k,i}$. Consequently, the feasible region of the trust-region subproblem is no longer a ball but rather an n-dimensional cylinder set defined by the projections

Algorithm 1: The UPOQA Algorithm

Input: Objective f, elements $\{f_i\}_{i=1}^q$, index sets $\{\mathcal{I}_i\}_{i=1}^q$, starting point x_{start}, initial trust-region resolution ρ_0
Output: Approximate solution to problem (1)

Initialize $\{\mathcal{Y}_{0,i}\}_{i=1}^q$ and $\{m_{0,i}\}_{i=1}^q$ centered at x_{start}, set $\boldsymbol{\Delta}_0 = [\rho_0, \ldots, \rho_0]^\top \in \mathbb{R}^q$;
Compute an improved starting point x_0 using the method described in Section 2.4;
for $k = 0, 1, \ldots$ **do**
  $F_1^{GI}, \ldots, F_q^{GI} \leftarrow$ **false**;
  Solve the trust-region subproblem

$$\min_{s \in \mathbb{R}^n} \quad \sum_{i=1}^q m_{k,i}(x_k + s),$$
$$\text{s.t.} \quad \|s^{\mathcal{I}_i}\|_2 \le \Delta_{k,i}, \quad \text{for } i = 1, 2, \ldots, q,$$

  to obtain the step s_k using the method detailed in Section 2.1 and set $\hat{x}_k \leftarrow x_k + s_k$;
  if $\max_i \left(\left\| s_k^{\mathcal{I}_i} \right\|_2 \right) \le \rho_k/2$ **then**
    for $i = 1, 2, \ldots, q$ **do**
      $\Delta_{k,i} \leftarrow \max\left(\Delta_{k,i}/2, \rho_k\right)$;
      if *the conditions for geometry improvement of model* $m_{k,i}$ *are met* **then**
        $F_i^{GI} \leftarrow$ **true**;
    if *entered this branch in multiple consecutive iterations* **or** $F_1^{GI}, , \ldots, F_q^{GI}$ *are all* ***false*** **then**
      $\rho_k \leftarrow 0.1\rho_k$;
  else
    Compute

$$r_k \leftarrow \frac{f(\hat{x}_k) - f(x_k)}{m_k(\hat{x}_k) - m_k(x_k)}, \quad r_{k,i} \leftarrow \frac{f_i(\hat{x}_k) - f_i(x_k)}{m_{k,i}(\hat{x}_k) - m_{k,i}(x_k)}, \quad \text{for } i = 1, 2, \ldots, q;$$

    for $i = 1, 2, \ldots, q$ **do**
      if $\hat{x}_k^{\mathcal{I}_i}$ *does not significantly deteriorate the poisedness of* $\mathcal{Y}_{k,i}$ **then**
        Select an interpolation point $y_k^{\text{del},i}$ to remove from $\mathcal{Y}_{k,i}$;
        Set $\mathcal{Y}_{k,i} \leftarrow \mathcal{Y}_{k,i} \setminus \left\{ y_k^{\text{del},i} \right\} \cup \hat{x}_k^{\mathcal{I}_i}$ and update the element model $m_{k,i}$;
      Adjust the trust-region radius $\Delta_{k,i}$ using the method detailed in Section 2.2;
    if $r_k \le 0.1$ **then**
      for $i = 1, 2, \ldots, q$ **do**
        if *the conditions for geometry improvement of model* $m_{k,i}$ *are met* **then**
          $F_i^{GI} \leftarrow$ **true**;
        if $F_1^{GI}, , \ldots, F_q^{GI}$ *are all* ***false*** **then**
          $\rho_k \leftarrow 0.1\rho_k$;
  Set x_{k+1} as the point with the smaller function value between x_k and $\hat{x}_k$;
  Perform geometry-improving steps for all element models and interpolation sets where $F_i^{GI} =$ **true**;

$\left\{P_{\mathcal{R}_i} \colon \mathbb{R}^n \to \mathcal{R}_i\right\}_{i=1}^q$ and the regions $\left\{\mathcal{B}_i(\Delta_{k,i}) \subset \mathcal{R}_i\right\}_{i=1}^q$:

$$\mathcal{S}(\Delta_k) \quad \triangleq \quad \bigcap_{i=1}^q P_{\mathcal{R}_i}^{-1}\left(\mathcal{B}_i\left(\Delta_{k,i}\right)\right) = \left\{ s \in \mathbb{R}^n \mid \|s^{\mathcal{I}_i}\| \le \Delta_{k,i},\ i = 1, 2, \ldots, q \right\}, \tag{5}$$

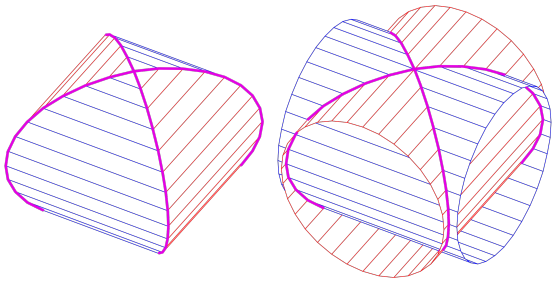


Fig. 1. Generation of the surface of a Steinmetz solid.

where $\boldsymbol{\Delta}_k = [\Delta_{k,1}, \ldots, \Delta_{k,q}]$. As before, $\|s^{\mathcal{I}_i}\|$ is shorthand for $\|P_{\mathcal{R}_i}(s)\|$. When the objective takes the form $f(x, y, z) = f_1(x, z) + f_2(y, z)$, $\|s^{\mathcal{I}_i}\|$ uses the L_2-norm, and $\Delta_{k,1} = \Delta_{k,2} \triangleq \overline{\Delta}$, the set $\mathcal{S}(\boldsymbol{\Delta}_k)$ will reduce to a Steinmetz solid, which is the intersection of two right circular cylinders with equal radii and orthogonal axes, as illustrated in Figure 1.

When $\Delta_{k,1} \neq \Delta_{k,2}$, the shape of $\mathcal{S}(\boldsymbol{\Delta}_k)$ becomes elongated, indicating that the model is more reliable in one direction than another, thus permitting larger steps in the reliable direction. In more general cases, the geometry of $\mathcal{S}(\boldsymbol{\Delta}_k)$ is highly complex, making the trust-region problem

$$\min_{s \in \mathcal{S}(\boldsymbol{\Delta}_k)} \quad m_k(x+s) = \sum_{i=1}^{q} m_{k,i}(x+s), \tag{6}$$

rather challenging to solve. It is straightforward to observe that the cylinder set $\mathcal{S}(\boldsymbol{\Delta}_k)$ contains an inscribed ball with radius $\Delta_{k,\min} \triangleq \min_{i=1,\ldots,q} \Delta_{k,i}$, denoted as $\mathcal{B}(\Delta_{k,\min})$. Thus, a cheap alternative is to solve the problem (6) approximately within $\mathcal{B}(\Delta_{k,\min})$. To address (6), Conn et al. [12] proposed a two-stage strategy. The first stage involves minimizing $m_k\left(x_k - t\nabla m_k(x_k)\right)$ within $\mathcal{B}(\Delta_{k,\min})$ to obtain a Cauchy step s_C. If further reduction can be achieved by increasing the step length, a subsequent minimization of $m_k\left(x_k - ts_C\right)$ is then attempted within $\mathcal{S}(\boldsymbol{\Delta}_k)$. In another DFO method `PSDFO` [11], all element models share the same trust-region radius, thereby circumventing the difficulty of handling structured trust regions. If derivatives are available, it is reasonable to reduce computational costs by simplifying the trust region geometry—for instance, replacing the original region with an inscribed ellipsoid or employing the L_∞-norm in the constraints to transform the trust region into a cube [55]. Nevertheless, to avoid potential degradation in subproblem solution accuracy caused by such simplifications, we prefer to maintain the trust region $\mathcal{S}(\boldsymbol{\Delta}_k)$ in its original form.

Since $\mathcal{S}(\boldsymbol{\Delta}_k)$ is the intersection of finitely many convex sets, it remains convex. To this end, we design an approximate projection operator $\widehat{P}_{\mathcal{S}(\boldsymbol{\Delta}_k)} : \mathbb{R}^n \to \mathcal{S}(\boldsymbol{\Delta}_k)$ and employ a modified projected gradient method to solve problem (6). The notation $P_{\mathcal{S}(\boldsymbol{\Delta}_k)}$ is reserved for the mathematically exact projection.

For projections onto convex sets formed by the intersection of multiple convex sets, a popular approach is to use the *averaged projection* [34]. Given arbitrary closed convex sets $F_1, \ldots, F_q$, the averaged projection is defined as

$$\overline{P}_{F_1,\ldots,F_q} \triangleq \frac{1}{q} \sum_{i=1}^{q} P_{F_i},$$

where P_{F_i} is the projection operator from $\mathbb{R}^n$ to F_i. For any point x_1 to be projected, one can iteratively compute $x_{k+1} = \overline{P}_{F_1,\ldots,F_q}(x_k)$ for $k = 1, 2, \ldots$ to obtain an approximation of the projection $P_{\mathcal{S}(\boldsymbol{\Delta}_k)}(x_1)$. Lewis et al. [34] proved that,

under a set of mild conditions on $F_1, \ldots, F_q$, this iteration reduces the mean squared distance $\sum_{i=1}^{q} d(x_k, F_i)^2/q$ at a linear rate of $1 - c/q$, where c is a constant depending on $F_1, \ldots, F_q$. This convergence rate implies that $O\left(\log \varepsilon / \log(1 - c/q)\right)$ iterations are sufficient to reduce the mean squared distance below any given tolerance $\varepsilon > 0$. Since $\log(1 - c/q) < -c/q$ always holds and $\log(1 - c/q)$ approaches $-c/q$ when q is large, the required number of iterations scales as $O(q \log(1/\varepsilon))$.

Each iteration of averaged projection involves recomputing all projections $P_{F_1}(x_k), \ldots, P_{F_q}(x_k)$. When F_i is a ball or a cylinder, computing $P_{F_i}(x_k)$ costs $O(n)$. Thus, the overall computational complexity of the averaged projection method is $O(nq^2 \log(1/\varepsilon))$.

Algorithm 2: Steinmetz Projection

Input: Point $s_0 \in \mathbb{R}^n$, index sets $\{\mathcal{I}_i\}_{i=1}^q$, trust-region radii $\{\Delta_i\}_{i=1}^q$
Output: An approximation of $P_{\mathcal{S}(\boldsymbol{\Delta})}(s_0)$

for $k = 0, 1, \ldots$ **do**
  Compute $v_i \leftarrow \left\| s_0^{\mathcal{I}_i} \right\|_2 / \Delta_i$ for $i = 1, \ldots, q$;
  if $\max_{i=1,\ldots,q} v_{k,i} \le 1$ **then**
    Break
  $\mathcal{G}_k \leftarrow \left\{ i = 1, \ldots, q \mid v_{k,i} = \max_{i=1,\ldots,q} v_{k,i} \right\}$;
  $s_k \leftarrow \texttt{SHRINK}\left(s_k,\ \{\mathcal{I}_i\}_{i=1}^q,\ \{\Delta_i\}_{i=1}^q,\ \mathcal{G}_k\right)$;

To improve the computational efficiency of projection operations, we propose an alternative approximate projection operator tailored for convex sets of the form $\mathcal{S}(\boldsymbol{\Delta})$, referred to as the *Steinmetz projection*. The procedure is outlined in Algorithm 2, which employs a subroutine SHRINK (see Algorithm 3). For clarity, we define the *violation ratio* of a point $s \in \mathbb{R}^n$ with respect to the i-th element as

$$v_i(s) \triangleq \frac{\left\| s^{\mathcal{I}_i} \right\|_2}{\Delta_i}.$$

The core idea of Steinmetz projection stems from an observation: Given an index set $\mathcal{G}$ and a point $s \in \mathbb{R}^n$ where each projection $s^{\mathcal{I}_i}$ satisfies $\|s^{\mathcal{I}_i}\| = \mu\Delta_i$ for all $i \in \mathcal{G}$, then scaling s by any real factor t yields $\|(ts)^{\mathcal{I}_i}\| = t\mu\Delta_i$ for all $i \in \mathcal{G}$. Thus, when projecting s onto $\mathcal{S}(\boldsymbol{\Delta})$, we first collect the indices of all elements with the highest violation ratios into a set $\mathcal{G}$, which by construction satisfies $\|s^{\mathcal{I}_i}\| = \mu\Delta_i$ for some $\mu \in \mathbb{R}$ and all $i \in \mathcal{G}$. Next, we rescale s in the subspaces corresponding to $\mathcal{G}$ until another element can be added into $\mathcal{G}$, repeating the process iteratively. Here, $\mathcal{G}$ is termed the *shrinking set*. Once $\mathcal{G}$ encompasses all elements and the current point s satisfies $\|s^{\mathcal{I}_i}\|_2 = \mu\Delta_i$ for all i, a final scaling by $1/\mu$ ensures the resulting point lies within $\mathcal{S}(\boldsymbol{\Delta})$. As shown in Algorithm 2, each iteration first identifies the shrinking set and then performs the shrinking operation.

The SHRINK subroutine (Algorithm 3) implements the shrinking operation. Assuming without loss of generality that the given point s lies outside $\mathcal{S}(\boldsymbol{\Delta})$, and given a shrinking set $\mathcal{G} = \{1, 2, \ldots, \hat{q}\}$ $(\hat{q} < q)$ satisfying

$$v_1(s) = \cdots = v_{\hat{q}}(s) > v_i(s), \quad \forall i \in \{\hat{q} + 1, \ldots, q\}, \tag{7}$$

the subroutine performs component-wise contraction in the subspace $\mathcal{R}_{\mathcal{G}} \triangleq \operatorname{span}\left(\cup_{i \in \mathcal{G}} \mathcal{R}_i\right)$. Specifically, it scales the component $s^{\mathcal{I}_{\mathcal{G}}}$ by a factor $t_* \in \mathbb{R}$, generating a new point $s_* \triangleq s + (t_* - 1)s^{\mathcal{I}_{\mathcal{G}}}$ that satisfies either:

$$1 = v_1(s_*) = \cdots = v_{\hat{q}}(s_*) \ge v_i(s_*), \quad \forall i \in \{\hat{q} + 1, \ldots, q\}, \tag{8}$$

Algorithm 3: Subroutine SHRINK

Input: Point $s \in \mathbb{R}^n$, index sets $\{\mathcal{I}_i\}_{i=1}^q$, trust-region radii $\{\Delta_i\}_{i=1}^q$, shrinking set $\mathcal{G}$
Output: Shrunk point s_*

1 Select arbitrary $i \in \mathcal{G}$ and compute $u \leftarrow \left\| s^{\mathcal{I}_i} \right\|_2 / \Delta_i$;
2 **if** $\mathcal{G} = \{1, 2, \ldots, q\}$ **then**
3 $\quad s_* \leftarrow s/u$;
4 **else**
5 $\quad \mathcal{I}_{\mathcal{G}} \leftarrow \bigcup_{i \in \mathcal{G}} \mathcal{I}_i$;
6 $\quad$ **for** $i = 1, \ldots, q$ **do**
7 $\quad\quad$ **if** $i \in \mathcal{G}$ **then**
8 $\quad\quad\quad t_i \leftarrow 1/u$;
9 $\quad\quad$ **else**
10 $\quad\quad\quad t_i \leftarrow \max\left\{ \frac{1}{u}, \frac{\|s^{\mathcal{I}_i \setminus \mathcal{I}_{\mathcal{G}}}\|_2}{\sqrt{u^2\Delta_i^2 - \|s^{\mathcal{I}_i \cap \mathcal{I}_{\mathcal{G}}}\|_2^2}} \right\}$;
11 $\quad t_* \leftarrow \max_{i=1,\ldots,q} t_i$;
12 $\quad s_* \leftarrow s + (t_* - 1)\, s^{\mathcal{I}_{\mathcal{G}}}$;

or introduces an elemental index $\hat{p} \in \{\hat{q}+1, \ldots, q\}$ such that

$$v_1(s_*) = \cdots = v_{\hat{q}}(s_*) = v_{\hat{p}}(s_*) \geq v_i(s_*), \quad \forall i \in \{\hat{q}+1, \ldots, q\} \setminus \{\hat{p}\}, \tag{9}$$

where $\mathcal{I}_{\mathcal{G}} \triangleq \bigcup_{i \in \mathcal{G}} \mathcal{I}_i$.

This mechanism enables the outer loop (Algorithm 2) to terminate the projection or subsequently expand the shrinking set to $\mathcal{G}_{\text{new}} = \{1, 2, \ldots, \hat{q}\} \cup \{\hat{p}\}$. The elemental scaling ratio t_i is the maximum of two possible choices, as computed in Steps 8 and 10 of Algorithm 3, yielding the candidate point $s_i \triangleq s + (t_i - 1)s^{\mathcal{I}_{\mathcal{G}}}$ satisfying either $1 = v_1(s_i) = \cdots = v_{\hat{q}}(s_i) > v_i(s_i)$ or $v_1(s_i) = \cdots = v_{\hat{q}}(s_i) = v_i(s_i)$. The final contraction ratio t_* is then selected as the maximum value of $t_1, \ldots, t_q$ ensuring satisfaction of (8) or (9). Note that the selection of trial index i in Step 1 is arbitrary, as the ratio $\left\| s^{\mathcal{I}_i} \right\|_2 / \Delta_i$ remains constant across all $i \in \mathcal{G}$ due to the defining property of the shrinking set $\mathcal{G}$. Theorem 2.1 formally establishes the correctness of the SHRINK subroutine.

Theorem 2.1. *Given a point $s \notin \mathcal{S}(\Delta)$, element index sets $\{\mathcal{I}_i\}_{i=1}^q$, trust-region radii $\{\Delta_i\}_{i=1}^q$, and a shrinking set $\mathcal{G}$ satisfying condition (7), there exists $\hat{p} \in \{\hat{q}+1, \ldots, q\}$ such that the point s_* returned by Algorithm 3 will satisfy either (8) or (9).*

Proof. Let $\hat{p} = \operatorname{argmax}_{i \notin \mathcal{G}} t_i$, where t_i is determined by Steps 8 and 10 of Algorithm 3. Let $u = \left\| s^{\mathcal{I}_i} \right\|_2 / \Delta_i$ for an arbitrary $i \in \mathcal{G}$, noting that this value is independent of the choice of i. Define $t_* = \max_{i=1,\ldots,q} t_i$ and $s_* = s + (t_* - 1)s^{\mathcal{I}_{\mathcal{G}}}$. We consider the size of $u^2\Delta_{\hat{p}}^2$ in two cases.

First, if $u^2\Delta_{\hat{p}}^2 > u^2 \left\| s^{\mathcal{I}_{\hat{p}} \setminus \mathcal{I}_{\mathcal{G}}} \right\|_2^2 + \left\| s^{\mathcal{I}_{\hat{p}} \cap \mathcal{I}_{\mathcal{G}}} \right\|_2^2$ holds, then we have

$$\frac{1}{u} > \frac{\|s^{\mathcal{I}_i \setminus \mathcal{I}_{\mathcal{G}}}\|_2}{\sqrt{u^2\Delta_i^2 - \|s^{\mathcal{I}_i \cap \mathcal{I}_{\mathcal{G}}}\|_2^2}},$$

and thus $t_{\hat{p}} = 1/u$. On one hand, $t_i = 1/u$ ($\forall i \in \mathcal{G}$), and on the other hand, $t_i \le t_{\hat{q}} = 1/u$ ($\forall i \notin \mathcal{G}$). Thus, $t_* = 1/u$. For any $i \in \mathcal{G}$, since $\mathcal{I}_i \subset \mathcal{I}_{\mathcal{G}}$, we have

$$s_*^{\mathcal{I}_i} = P_{\mathcal{R}_i} s_* = t_* s^{\mathcal{I}_i}, \quad \forall i \in \mathcal{G}. \tag{10}$$

Furthermore, by the definition of u, it follows that

$$s_*^{\mathcal{I}_i} = \frac{1}{u} s^{\mathcal{I}_i} \Longrightarrow \left\| s_*^{\mathcal{I}_i} \right\|_2 = \frac{1}{u} \left\| s^{\mathcal{I}_i} \right\|_2 = \Delta_i \Longrightarrow v_i(s_*) = 1, \quad \forall i \in \mathcal{G}.$$

Meanwhile, for any $i \notin \mathcal{G}$, since

$$s_*^{\mathcal{I}_i} = s^{\mathcal{I}_i \setminus \mathcal{I}_{\mathcal{G}}} + t_* s^{\mathcal{I}_i \cap \mathcal{I}_{\mathcal{G}}}, \quad \forall i \notin \mathcal{G}, \tag{11}$$

and $s^{\mathcal{I}_i \setminus \mathcal{I}_{\mathcal{G}}} \perp s^{\mathcal{I}_i \cap \mathcal{I}_{\mathcal{G}}}$, we have

$$\left\| s_*^{\mathcal{I}_i} \right\|_2^2 = \left\| s^{\mathcal{I}_i \setminus \mathcal{I}_{\mathcal{G}}} \right\|_2^2 + \frac{1}{u^2} \left\| s^{\mathcal{I}_i \cap \mathcal{I}_{\mathcal{G}}} \right\|_2^2 \le \Delta_i^2 \Longrightarrow v_i(s_*) \le 1, \quad \forall i \notin \mathcal{G}.$$

Thus, in this case, inequality (8) holds.

In the second case, when $u^2 \Delta_{\hat{p}}^2 \le u^2 \left\| s^{\mathcal{I}_{\hat{p}} \setminus \mathcal{I}_{\mathcal{G}}} \right\|_2^2 + \left\| s^{\mathcal{I}_{\hat{p}} \cap \mathcal{I}_{\mathcal{G}}} \right\|_2^2$ holds, we have $t_{\hat{p}} \ge 1/u$, and thus $t_* = t_{\hat{p}}$. For any $i \in \mathcal{G}$, since equation (10) still holds, it follows that $v_i(s_*) = t_* v_i(s) = t_{\hat{p}} u$. Therefore, all $v_i(s_*)$ ($\forall i \in \mathcal{G}$) remain the same. From equation (11), we derive

$$\left\| s_*^{\mathcal{I}_{\hat{p}}} \right\|_2^2 = \left\| s^{\mathcal{I}_{\hat{p}} \setminus \mathcal{I}_{\mathcal{G}}} \right\|_2^2 + t_{\hat{p}}^2 \left\| s^{\mathcal{I}_{\hat{p}} \cap \mathcal{I}_{\mathcal{G}}} \right\|_2^2 = \left\| s^{\mathcal{I}_{\hat{p}} \setminus \mathcal{I}_{\mathcal{G}}} \right\|_2^2 + \frac{\left\| s^{\mathcal{I}_{\hat{p}} \setminus \mathcal{I}_{\mathcal{G}}} \right\|_2^2 \left\| s^{\mathcal{I}_{\hat{p}} \cap \mathcal{I}_{\mathcal{G}}} \right\|_2^2}{u^2 \Delta_{\hat{p}}^2 - \left\| s^{\mathcal{I}_{\hat{p}} \cap \mathcal{I}_{\mathcal{G}}} \right\|_2^2} = \frac{u^2 \Delta_{\hat{p}}^2 \left\| s^{\mathcal{I}_{\hat{p}} \setminus \mathcal{I}_{\mathcal{G}}} \right\|_2^2}{u^2 \Delta_{\hat{p}}^2 - \left\| s^{\mathcal{I}_{\hat{p}} \cap \mathcal{I}_{\mathcal{G}}} \right\|_2^2} = u^2 \Delta_{\hat{p}}^2 t_{\hat{p}}^2.$$

Thus, we have $v_{\hat{p}}(s_*) = t_{\hat{p}} u$. For any $i \notin \mathcal{G}$, we claim that $v_i(s_*) \le t_{\hat{p}} u$. Otherwise, the inequality $\left\| s_*^{\mathcal{I}_i} \right\|_2 > t_{\hat{p}} u \Delta_i$ would hold, leading to

$$\left\| s^{\mathcal{I}_i \setminus \mathcal{I}_{\mathcal{G}}} \right\|_2^2 + t_{\hat{p}}^2 \left\| s^{\mathcal{I}_i \cap \mathcal{I}_{\mathcal{G}}} \right\|_2^2 > t_{\hat{p}}^2 u^2 \Delta_i^2 \Longrightarrow \frac{\left\| s^{\mathcal{I}_i \setminus \mathcal{I}_{\mathcal{G}}} \right\|_2}{\sqrt{u^2 \Delta_i^2 - \left\| s^{\mathcal{I}_i \cap \mathcal{I}_{\mathcal{G}}} \right\|_2^2}} > t_{\hat{p}},$$

which implies $t_i > t_{\hat{p}}$. This contradicts the assumption $\hat{p} = \operatorname{argmax}_{i \notin \mathcal{G}} t_i$. Therefore, we have

$$t_{\hat{p}} u = v_j(s_*) = v_{\hat{p}}(s_*) \ge v_i(s_*), \quad \forall\, i \in \mathcal{G}^c \setminus \{\hat{p}\},\ j \in \mathcal{G}.$$

Hence, inequality (9) holds. □

Note that at the initialization of Steinmetz projection, we may exclude the index sets $\mathcal{I}_j$ of elements satisfying $v_j(s_0) \le 1$ from $\{\mathcal{I}_i\}_{i=1}^q$, since they never contribute t_j values exceeding $1/u$ and thus cannot affect the value of t_*. Furthermore, Steps 2 and 5 of Algorithm 2 need not be explicitly recomputed in each iteration, as the new shrinking set can always be obtained simply using intermediate results from the subroutine SHRINK. If $\left|\left\{i \mid v_j(s_0) \le 1\right\}\right| = \overline{q}$, then each run of Algorithm 3 has a cost of $O\left(n\left(q - \overline{q} - |\mathcal{G}|\right)\right)$. Since each run either terminates the projection or adds at least one new member to the shrinking set, the algorithm iterates at most $q - \overline{q}$ times. Therefore, the computational complexity of Algorithm 2 is

$$O\left(\sum_{i=1}^{q-\overline{q}} n(q - \overline{q} - i)\right) = O\left(\frac{1}{2} n (q - \overline{q})^2\right). \tag{12}$$

Meanwhile, the averaged projection method with stopping tolerance ε has a cost of $O(n(q-\overline{q})^2 \log \varepsilon)$. By contrast, the Steinmetz projection does not require setting termination precision, resulting in a lower constant factor. Figure 2

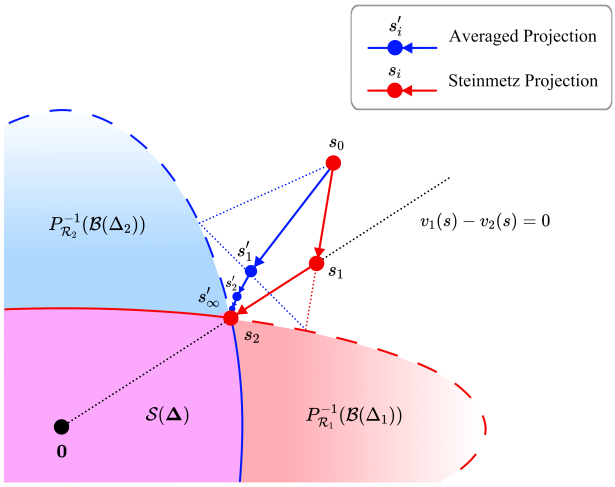


Fig. 2. Iterates generated by applying the averaged projection and the Steinmetz projection to the initial point s_0, where the marked $\mathcal{S}(\boldsymbol{\Delta})$, $P_{\mathcal{R}_1}^{-1}(\mathcal{B}(\Delta_1))$, and $P_{\mathcal{R}_2}^{-1}(\mathcal{B}(\Delta_2))$ represent their cross-sections in the current region.

illustrates the iterates generated by applying both averaged projection and Steinmetz projection to the initial point s_0, where the labeled $\mathcal{S}(\boldsymbol{\Delta})$, $P_{\mathcal{R}_1}^{-1}(\mathcal{B}(\Delta_1))$, and $P_{\mathcal{R}_2}^{-1}(\mathcal{B}(\Delta_2))$ represent their cross-sections in the current region.

It should be noted that neither the Steinmetz projection nor the averaged projection method constitutes a projection in the mathematical sense (i.e., finding the closest point under a given norm), and it is difficult to assert which method consistently yields better optimality for the projected points. Based on our numerical tests, we found that first applying the averaged projection for $k_{\text{avg}} = 1 \sim 5$ iterations, followed by the Steinmetz projection, sometimes produces a point closer to the initial point than using either method alone. For any point $s \in \mathbb{R}^n$ to be projected, we denote the resulting projected point obtained through this combined approach as $\widehat{P}_{\mathcal{S}(\boldsymbol{\Delta})}(s;\ k_{\text{avg}})$.

To solve the trust-region subproblem (6), we employ a modified projected gradient method (Algorithm 4), which closely follows the truncated conjugate gradient method. The key difference arises when the iterate $s_k + \alpha_k p_k$ falls outside $\mathcal{S}(\boldsymbol{\Delta})$. Instead of stopping, the algorithm performs an exact line search along the direction $\widehat{P}_{\mathcal{S}(\boldsymbol{\Delta})}(s_k + \alpha_k p_k;\ k_{\text{avg}}) - s_k$ starting from s_k to determine s_{k+1}. Since $\widehat{P}_{\mathcal{S}(\boldsymbol{\Delta})}(\cdot;\ k_{\text{avg}})$ is not an exact projection, monotonic decrease in the function values cannot be guaranteed, making this additional line search necessary. Furthermore, the algorithm alternates between $k_{\text{avg}}^{(1)} = 0$ and $k_{\text{avg}}^{(2)} = 4$ during iterations to mitigate the limitations of using a single projection operator. The truncation radius $\widehat{\Delta}$ in Algorithm 4 is chosen sufficiently large to ensure that the trust region is entirely contained within the ball $\mathcal{B}(\widehat{\Delta})$. In addition, a comparison of UPOQA's performance with versus without structured trust regions is provided in Appendix B, where the non-structured version employs a spherical trust region, and the resulting subproblem is solved via the truncated conjugate gradient method with a boundary improvement step.

2.2 Managment of Trust Region Radii

In Step 19 of Algorithm 1, we employ the combined separation criterion proposed by Shahabuddin [55, Section 2.4] to adjust the trust region radii $\{\Delta_{k,i}\}_{i=1}^{q}$. Here we briefly introduce this criterion.

Algorithm 4: Modified Projected Gradient Method for the Trust-Region Subproblem (6)

Input: Overall model m, center point x, truncation radius $\widehat{\Delta} > 0$, projection operator $\widehat{P}_{\mathcal{S}(\Delta)}$, numbers of start-up averaged projection iterations $k_{\text{avg}}^{(1)},\ k_{\text{avg}}^{(2)} \geq 0$

Output: An approximate solution to (6)

Set the iterate $s_0 \leftarrow 0$;
Set the search direction $p_0 \leftarrow -\nabla m(x + s_0)$;
for $k = 0, 1, \ldots$ **until** $\nabla m(x + s_k)^\top p_k \geq 0$ **do**
  Compute $\alpha_k \leftarrow \operatorname{argmin} \left\{ m(x + s_k + \alpha p_k) \mid 0 \leq \alpha \leq \widehat{\Delta} / \|p_k\|_2 \right\}$;
  if $s_k + \alpha^k p_k$ *falls outside* $\mathcal{S}(\Delta)$ **then**
    $\hat{s}_k \leftarrow \widehat{P}_{\mathcal{S}(\Delta)} \left(s_k + \alpha_k p_k;\ k_{\text{avg}}^{(k \bmod 2)} \right)$;
    $\hat{p}_k \leftarrow \hat{s}_k - s_k$;
    $\hat{\alpha}_k \leftarrow \operatorname{argmin} \{ m(x + s_k + \alpha \hat{p}_k) \mid 0 \leq \alpha \leq 1 \}$;
    $s_{k+1} \leftarrow s_k + \hat{\alpha}_k \hat{p}_k$;
    $p_{k+1} \leftarrow -\nabla m(x + s_{k+1})$;
  else
    $s_{k+1} \leftarrow s_k + \alpha_k p_k$;
    $\beta_k \leftarrow \|\nabla m(x + s_{k+1})\|^2 / \|\nabla m(x + s_k)\|^2$;
    $p_{k+1} \leftarrow -\nabla m(x + s_{k+1}) + \beta_k p_k$;

In trust region methods, it is common practice to adjust the trust region based on the value of the reduction ratio $r_k = \delta_k f / \delta_k m_k$, where

$$\delta_k f = f(x_k) - f(\hat{x}_k), \quad \delta_k m_k = m_k(x_k) - m_k(\hat{x}_k).$$

However, when solving partially separable problems, it is infeasible to independently adjust each $\Delta_{k,i}$ solely based on $r_{k,i}$. For an individual objective, we always expect the surrogate model and the objective function to decrease. However, for coexisting element functions, the potential antagonistic relationship between them may necessitate that we sometimes tolerate a slight deterioration in the function values of some elements. This suggests that the algorithm should not simply encourage larger $r_{k,i}$ values. Moreover, the cancellation issue further complicates the problem. Conn et al. [12] first pointed out that a situation may arise where $\delta_k m_{k,1}, \ldots, \delta_k m_{k,q}$ have large absolute magnitudes, but cancellation occurs, resulting in a very small absolute value for the aggregated $\delta_k m_k$. In such cases, rounding errors in $\delta_k m_{k,1}, \ldots, \delta_k m_{k,q}$ can easily lead to inaccurate computation of $\delta_k m_k$ and to misjudgment of the reliability of the overall model m_k.

For above considerations, Shahabuddin [55, Section 2.4] proposed a scoring-based strategy. The scores consist of a global score τ_k and element-wise local scores $\{\hat{\tau}_{k,i}\}_{i=1}^q$, both taking integer values of 0, 1, or 2. By summing the two parts, the total score (ranging from 0 to 4) determines how each element's trust region radius $\Delta_{k,i}$ should be adjusted. The global score τ_k depends solely on the magnitude of r_k—the larger r_k, the higher τ_k. The element-wise score $\hat{\tau}_{k,i}$ is determined by the position of the point $(\delta_k m_{k,i}, \delta_k f_i)$. Figure 3 illustrates the acceptable region for assigning $\{\hat{\tau}_{k,i}\}_{i=1}^q$, bounded by two rays with different slopes emanating from the origin and a line with slope 1 that does not pass through the origin. The stricter the acceptable region in which $(\delta_k m_{k,i}, \delta_k f_i)$ falls, the higher $\hat{\tau}_{k,i}$. The detailed scoring strategy is described in Algorithm 5. After scoring, we set $\tau_{k,i} = \tau_k + \hat{\tau}_{k,i}$. The new trust region radii are then updated as follows:

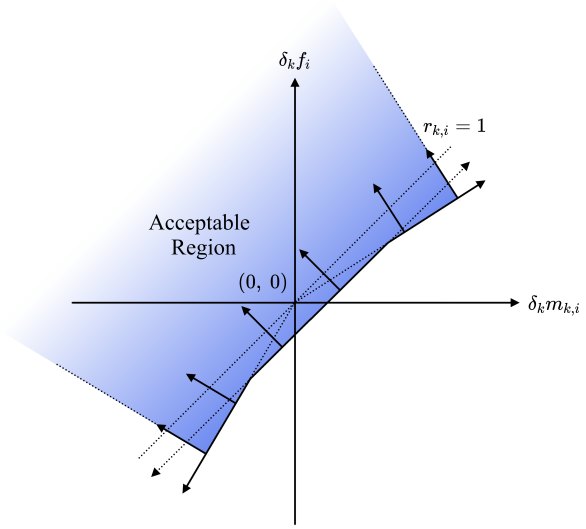


Fig. 3. The combined separation criterion proposed by Shahabuddin [55, Section 2.4] determines whether to expand the elemental trust region radius $\Delta_{k,i}$ based on the slope $r_{k,i}$ of the point $(\delta_k m_{k,i},\ \delta_k f_i)$ relative to the origin and its distance to the line $r_{k,i} = 1$. The colored region in the figure represents the acceptable region for expansion operations.

$$\Delta_{k+1,i} \begin{cases} \in [1,\ \theta_4]\Delta_{k,i}, & \text{if } \tau_{k,i} = 4, \\ \in [1,\ \theta_3]\Delta_{k,i}, & \text{if } \tau_{k,i} = 3, \\ \in [\theta_2,\ 1]\Delta_{k,i}, & \text{if } \tau_{k,i} = 2, \\ = \theta_2\Delta_{k,i}, & \text{if } \tau_{k,i} = 1, \\ = \theta_1\Delta_{k,i}, & \text{if } \tau_{k,i} = 0, \end{cases}$$

In our implementation, the parameters are set as $\theta_1 = 0.5$, $\theta_2 = 1/\sqrt{2}$, $\theta_3 = \sqrt{2}$, and $\theta_4 = 2$. Additionally, Step 21 in Algorithm 5 ensures that when the overall model reduction is unsatisfactory ($r_k < \mu_1$), at least one element's trust region radius undergoes substantial decrease to prevent the algorithm from stagnating at a fixed radius. Although such cases are rare, they have indeed been observed in our numerical experiments. This step is the only modification we made to Shahabuddin's combined separation criterion [55, Section 2.4].

2.3 Selective Updates of Element Models

After obtaining the solution s_k to the trust-region subproblem (6), it is time to update element models and interpolation sets. For the model $m_{k,i}$, the new interpolation point generated by the solution s_k is $\hat{x}_k^{\mathcal{I}_i} = (x_k + s_k)^{\mathcal{I}_i}$. However, blindly adding $\hat{x}_k^{\mathcal{I}_i}$ to the set $\mathcal{Y}_{k,i}$ may deteriorate the poisedness of $\mathcal{Y}_{k,i}$. For example, when the objective function is $f(x, y) = f_1(x) + f_2(y)$ and the starting point is (x_0, y_0), if x_0 is already very close to a local minimizer of f_1, the computed trust-region step will have vastly different scales in the subspaces corresponding to x and y. This can cause a drastic contraction in the geometry of the interpolation set $\mathcal{Y}_1$, potentially leading to numerical instability. If the algorithm detects such a risk, it skips such updates, as implemented in Step 16 of Algorithm 1.

Algorithm 5: Scoring Strategy for Trust Region Radius Adjustment

Input: Elemental model reductions $\{\delta_k m_{k,i}\}_{i=1}^q$, elemental reduction ratios $\{r_{k,i}\}_{i=1}^q$, trust region radii $\{\Delta_{k,i}\}_{i=1}^q$, trust region resolution ρ_k, overall reduction ratio r_k, constants $0 < \mu_1 < \mu_2 < 1$

Output: Total scores for each element $\{\tau_{k,i}\}_{i=1}^q$

$\zeta_k \leftarrow \dfrac{\sum_{\delta_k m_{k,i}<0} \delta_k m_{k,i}}{\sum_{\delta_k m_{k,i}\geq 0} \delta_k m_{k,i}}$;

For $i = 1, 2$, set $\eta_i = -(1-\mu_i)\zeta_k$, $\alpha_i \leftarrow \dfrac{(\mu_i+\eta_i)(1+\zeta_k) - 2\zeta_k}{1-\zeta_k}$;

if $r_k \geq \mu_2$ **then** $\tau_k \leftarrow 2$;

else if $r_k \geq \mu_1$ **then** $\tau_k \leftarrow 1$;

else $\tau_k \leftarrow 0$;

for $i = 1, 2, \ldots, q$ **do**

  if $\delta_k m_{k,i} \geq 0$ **then**

    if $r_{k,i} \geq \alpha_2$ **or** $\delta_k f_i \geq \delta_k m_{k,i} - \eta_2(\delta_k m_k)/q$ **then**

      $\hat{\tau}_{k,i} \leftarrow 2$;

    else if $r_{k,i} \geq \alpha_1$ **or** $\delta_k f_i \geq \delta_k m_{k,i} - \eta_1(\delta_k m_k)/q$ **then**

      $\hat{\tau}_{k,i} \leftarrow 1$;

    else $\hat{\tau}_{k,i} \leftarrow 0$;

  else

    if $r_{k,i} \leq 2 - \alpha_2$ **or** $\delta_k f_i \geq \delta_k m_{k,i} - \eta_2(\delta_k m_k)/q$ **then**

      $\hat{\tau}_{k,i} \leftarrow 2$;

    else if $r_{k,i} \leq 2 - \alpha_1$ **or** $\delta_k f_i \geq \delta_k m_{k,i} - \eta_1(\delta_k m_k)/q$ **then**

      $\hat{\tau}_{k,i} \leftarrow 1$;

    else $\hat{\tau}_{k,i} \leftarrow 0$;

  $\tau_{k,i} \leftarrow \hat{\tau}_{k,i} + \tau_k$;

if $\tau_k = 0$ **then**

  Verify all $\tau_{k,i}$ to ensure at least one element satisfies $\tau_{k,i} \leq 1$ and $\Delta_{k,i} > \rho_k$. Otherwise, set $\tau_{k,i}$ of the lowest-scoring element with $\Delta_{k,i} > \rho_k$ to 0;

In Powell's DFO methods that based on quadratic models, the denominator σ in the update formula for the interpolation system is a crucial parameter [45]. Its magnitude largely determines the impact of the update on the poisedness of the interpolation system, and a larger value is preferred. Therefore, we decide whether to accept the new interpolation point $\hat{x}_k^{\mathcal{I}_i}$ into $\mathcal{Y}_{k,i}$ by checking the value $\sigma_{k,i}$ during the update of $m_{k,i}$. We first compute $\sigma_{k,i}$ and multiply it by a factor $\gamma_{k,i} \triangleq \min\left\{\left\|s_k^{\mathcal{I}_i}\right\|_2 / \rho_k,\ 1\right\}$, where s_k is the solution to the trust-region subproblem (6) solved in Step 5 of Algorithm 1. The factor $\gamma_{k,i}$ penalizes situations with small step updates, as they generally contribute limited improvements to the model's accuracy, even if they don't significantly compromise the system's poisedness. Due to rounding errors, the value of $\sigma_{k,i}$ may become negative. To mitigate this, we check the intermediate variables used in computing $\sigma_{k,i}$ to detect such tendencies, and apply an additional penalty factor $w_{k,i}$ to the value of $\sigma_{k,i}$. This penalty factor is determined heuristically - in short, we reduce positive $\sigma_{k,i}$ values while enlarging negative ones. The resulting value $w_{k,i}\gamma_{k,i}\sigma_{k,i}$ is then compared with a fixed threshold ξ. The new interpolation point $\hat{x}_k^{\mathcal{I}_i}$ is accepted only if $w_{k,i}\gamma_{k,i}\sigma_{k,i}$ exceeds the threshold. In `UPOQA`, the default threshold is set to $\xi = 10^{-5}$.

In most cases, updates with negative $\sigma_{k,i}$ are automatically rejected. However, an exception occurs in extreme situations where all candidate updates yield negative $\sigma_{k,i}$ values, and the algorithm will accept the update whose $w_{k,i}\gamma_{k,i}\sigma_{k,i}$ is closest to zero. Only then does the additional penalty on negative $\sigma_{k,i}$ values matter.

2.4 Starting Point Search

As mentioned earlier, based on the element values evaluated at a certain number of interpolation points selected from the grid

$$\left\{ x = \sum_{i=1}^{n} c_i e_i \mid c_1, \ldots, c_n \in \mathbb{Z} \right\},$$

one can easily compute significantly more overall function values. This idea has been adopted by Price and Toint [49]. Although UPOQA cannot directly utilize this search pattern to accelerate the algorithm, it employs similar strategy to select a better starting point for the algorithm.

UPOQA initializes interpolation sets in the same way as COBYQA [50]. Suppose $\mathcal{Y}_0 = \{y_0^1, \ldots, y_0^N\}$ is the n-dimensional interpolation set to be initialized with a capacity of $N = 2n+1$, the initial trust-region radius is Δ_0, and the user-specified starting point is x_{start}. Then, the points $y_0^1, \ldots, y_0^N$ are initialized as

$$y_0^i \triangleq \begin{cases} x_{\text{start}}, & \text{if } \ i = 1, \\ x_{\text{start}} + \Delta_0 e_{i-1}, & \text{if } \ 2 \le i \le n+1, \\ x_{\text{start}} - \Delta_0 e_{i-n-1}, & \text{if } \ n+2 \le i \le 2n+1. \end{cases}$$

Similarly, for each elemental interpolation set $\mathcal{Y}_{0,i}$, UPOQA initializes it with $x_{\text{start}}^{\mathcal{I}_i}$ as the center and Δ_0 as the radius. Based on the resulting sets $\{\mathcal{Y}_{0,i}\}_{i=1}^{q}$, UPOQA then explores a certain number of solutions of a constraint satisfaction problem (CSP) using the minimum remaining values heuristic. In this CSP, the state is a complete point $x \in \mathbb{R}^n$, and the constraints require that for any $i = 1, \ldots, q$, $P_{\mathcal{R}_i}(x) \in \mathcal{Y}_{0,i}$ must hold. Since the starting point search mechanism has limited impact on improving algorithmic performance, we restrict the search to at most 5000 solutions and select the point with the smallest objective function value as the actual starting point x_0 for UPOQA.

2.5 Optional Features

Besides the procedure demonstrated in Algorithm 1, UPOQA also provides two additional functionalities to enhance its flexibility and applicability: restart mechanism and hybrid black-white-box optimization.

2.5.1 Restart Mechanism. In model-based derivative-free methods, the trust region radius Δ or trust region resolution ρ decreasing to a minimal value often indicates algorithm convergence. However, when the objective function contains noise, the noise will dominate the function value fluctuations once Δ becomes sufficiently small. Consequently, the interpolation model will no longer approximate the objective but model the noise instead. This leads to ineffective descent steps and may even trigger numerical instabilities causing algorithm failure.

Restart mechanisms have been widely adopted in other areas of numerical optimization, such as the conjugate gradient method and GMRES. Cartis et al.'s Py-BOBYQA [7] introduced this mechanism into model-based DFO methods, with experiments demonstrating its robustness for noise. Inspired by this, UPOQA incorporates the soft restart mechanism similar to that in Py-BOBYQA. This mechanism is only activated upon user request. When numerical singularity or trust-region resolution depletion occurs, the algorithm resets Δ and ρ to larger values, selectively replaces some existing

interpolation points with a few randomly generated ones, and relocates the iterate to a distant position before resuming execution.

2.5.2 Hybrid Black-White-Box Optimization. Real-world partially separable problems may exhibit more complex structures than (1). To maximize the algorithm's applicability, we adapt the implementation of UPOQA to handle problems of the form

$$\min_{x\in\mathbb{R}^n} \; f(x) = f_0(x) + \sum_{i=1}^{q} w_i h_i\left(f_i(x)\right), \tag{13}$$

where $f_1, \ldots, f_q$ remain black-box functions as before, while $f_0 : \mathbb{R}^n \to \mathbb{R}$ is a white-box function whose gradient and Hessian can be calculated directly. Each $h_i : \mathbb{R} \to \mathbb{R}$ is a transformation applied to the element value $f_i(x)$, also treated as white-box, and $w_i \in \mathbb{R}$ denotes the weight for each element. The transition from (1) to (13) requires minimal algorithmic modifications, as the added white-box components can be directly incorporated into the trust-region subproblem construction, yet it significantly enhances flexibility. Moreover, UPOQA allows users to dynamically update all $w_1, \ldots, w_q$ and $h_1, \ldots, h_q$ via a callback function during execution. This enables real-time trade-offs among multiple elements in multi-objective optimization, as well as effortless adjustment of penalty coefficients in constrained optimization without manual restarting.

An additional advantage of this adaptation stems from UPOQA's use of quadratic models for each black-box f_i. If f_i inherently and locally resembles a quadratic function, maintaining an accurate surrogate model becomes computationally inexpensive, reducing unnecessary trial-and-error during optimization. Hence, knowledgeable users may preemptively extract known non-quadratic composite parts from f_i and pass them as h_i to the algorithm. This strategy lowers the cost of maintaining interpolation models and ultimately decreases the number of function evaluations required for convergence.

3 NUMERICAL EXPERIMENTS

We briefly describe our testing methodology, the preparation of test problems, and the results of numerical experiments. Our primary evaluation metrics are the performance profile and data profile proposed by Moré and Wild [38], supplemented with an additional speed-up profile to assess the acceleration effect of UPOQA when exploiting partially separable structures. The UPOQA algorithm has been implemented as a Python library named upoqa. We conducted tests on benchmark problems extracted from CUTEst [20] using the S2MPJ [21] tool, along with a set of quantum variational problems. All tests were conducted on a server equipped with a 32-core Hygon C86 7285 CPU and 504 GB of RAM.

3.1 Testing Methodology

Let $\mathcal{S}$ denote the set of all solvers participating in the test, and $\mathcal{P}$ denote the set of test problems used. Each test problem $p \in \mathcal{P}$ has a fixed starting point $x_{0,p}$ and an objective function f_p. Given a tolerance $\varepsilon \in (0, 1)$, a solver $s \in \mathcal{S}$ is said to converge on problem p with tolerance ε if and only if its iterate x satisfies

$$f_p(x) \le f_p^* + \varepsilon \left[f_p(x_{0,p}) - f_p^* \right], \tag{14}$$

where f_p^* is the lowest function value achieved by any solver in $\mathcal{S}$ on problem p. Let $t_{p,s}$ denote the number of function evaluations required for solver s to converge on problem p. If convergence is not achieved, we set $t_{p,s} = \infty$.

The *performance profile* [38] is a concept introduced to evaluate and compare different optimization algorithms. This metric assesses which solver has an advantage in computational cost for given test problems. First, the performance

ratio is defined as

$$r_{p,s} \triangleq \frac{t_{p,s}}{\min_{u \in \mathcal{S}}\{t_{p,u}\}},$$

which can be viewed as a relative cost incurred by solver s to solve problem p. Then, the performance profile is defined as

$$\rho_s(\alpha) \triangleq \frac{1}{\operatorname{card}(\mathcal{P})} \operatorname{card}\left(\left\{p \in \mathcal{P} : r_{p,s} \leq \alpha\right\}\right), \quad \text{for } \alpha \geq 1,$$

where $\operatorname{card}(\cdot)$ denotes the cardinality of a set. The metric $\rho_s(\alpha)$ can be interpreted as the proportion of problems in $\mathcal{P}$ that solver s can solve with a performance ratio not exceeding α. Specifically, $\rho_s(1)$ represents the proportion of problems where solver s converges faster than all other solvers, while $\rho_s(+\infty)$ indicates the proportion of problems that s can solve regardless of cost. Given two solvers s_1 and s_2, $\rho_{s_1}(\alpha) > \rho_{s_2}(\alpha)$ implies that under the constraints $r_{p,s_1} \leq \alpha$ and $r_{p,s_2} \leq \alpha$, s_1 can successfully solve more problems than s_2. The value of $\rho_s(\alpha)$ for an individual solver is highly dependent on the competitors present in $\mathcal{S}$.

In contrast, the *data profile* [38] focuses more on the absolute capability of a single solver and is defined as

$$d_s(\alpha) \triangleq \frac{1}{\operatorname{card}(\mathcal{P})} \operatorname{card}\left(\left\{p \in \mathcal{P} : \frac{t_{p,s}}{n_p + 1} \leq \alpha\right\}\right), \quad \text{for } \alpha \geq 0,$$

where n_p denotes the dimension of problem p. The metric $d_s(\alpha)$ can be interpreted as the proportion of problems in $\mathcal{P}$ that solver s can solve using no more than $\alpha(n_p + 1)$ function evaluations. Specifically, $d_s(0) = 0$, while $d_s(+\infty)$ represents the proportion of problems that s can solve regardless of computational cost. Given two solvers s_1 and s_2, $d_{s_1}(\alpha) > d_{s_2}(\alpha)$ implies that, for each problem p, when using no more than $\alpha(n_p + 1)$ function evaluations, s_1 can successfully solve more problems than s_2.

When UPOQA converges on a given problem, the number of function evaluations for each element may differ. This discrepancy may arise from variations in the number of interpolation points required to initialize the surrogate models or from whether geometry improvement steps are performed on each element model during iterations. Let $t_{p,s}^{(i)}$ denote the number of function evaluations required for element f_i when solver s converges on problem p, $t_{p,s}^{\text{wst}}$ the maximum among $t_{p,s}^{(1)}, \ldots, t_{p,s}^{(q)}$, and $t_{p,s}^{\text{avg}}$ their average. If the cost of evaluating the objective function is dominated by a specific element f_{i_*}, the overall cost of the algorithm will be determined by $t_{p,s}^{(i_*)}$. Hence, in the most pessimistic scenario, the cost should be measured by $t_{p,s}^{\text{wst}}$ rather than $t_{p,s}^{\text{avg}}$. This is the meaning of the superscript wst (worst-case). Unless otherwise specified, the numerical experiments in subsequent sections generally use $t_{p,s}^{\text{wst}}$ to quantify the computational cost of UPOQA. When no ambiguity arises, $t_{p,s}^{\text{wst}}$ may be abbreviated as t_p^{wst}.

Since UPOQA leveraging partially separable structures is often significantly faster than algorithms that do not utilize such structures—potentially rendering comparisons between the two less meaningful—this paper introduces an alternative metric called the *speed-up profile*, defined as

$$\operatorname{su}(\alpha) \triangleq \begin{cases} \frac{1}{\operatorname{card}(\mathcal{P})} \operatorname{card}\left(\left\{p \in \mathcal{P} : 1 \leq c_p \leq \alpha \wedge \left(t_p^{\text{single}} < \infty \vee t_p^{\text{wst}} < \infty\right)\right\}\right), & \text{for } \alpha \geq 1, \\ \frac{1}{\operatorname{card}(\mathcal{P})} \operatorname{card}\left(\left\{p \in \mathcal{P} : \alpha \leq c_p < 1 \wedge \left(t_p^{\text{single}} < \infty \vee t_p^{\text{wst}} < \infty\right)\right\}\right), & \text{for } 0 \leq \alpha < 1, \end{cases}$$

where

$$c_p \triangleq \frac{t_p^{\text{single}} / t_p^{\text{wst}}}{n_p \,/ \max\limits_{i=1,\ldots,q_p} n_{p,i}} \tag{15}$$

and t_p^{single} represents the number of function evaluations needed when the objective is treated as a single element and passed to UPOQA. Here, $n_{p,i}$ is the dimension of the element f_i for problem p. Thus, $t_p^{\text{single}} / t_p^{\text{wst}}$ measures the

actual speed-up achieved by exploiting the partially separable structure, while $n_p / \max_{i=1,\ldots,q_p} n_{p,i}$ serves as a predicted speed-up ratio. To illustrate, consider an n-dimensional objective function formed by summing $n/\hat{n}$ low-dimensional subfunctions, each depending on distinct $\hat{n}$ variables. The Hessian is block-diagonal with identical $\hat{n} \times \hat{n}$ blocks. For gradient-based methods using forward-difference numerical gradients, exploiting this structure reduces per-iteration function evaluations from $O(n)$ to $O(\hat{n})$—each subgradient requires $\hat{n} + 1$ evaluations rather than $n + 1$ for the full gradient. Assuming comparable iteration counts, this induces a predicted speed-up ratio of $n/\hat{n}$. For general problems, the elemental dimensions $n_1, \ldots, n_q$ may vary. In such cases, we assume that the element with largest dimension dominates the cost, making $n / \max_{i=1,\ldots,q_p} n_i$ a conservative prediction of the speed-up ratio. Notably, if any element has the same dimension as the objective function (i.e., $\max_{i=1,\ldots,q_p} n_i = n$), `UPOQA` may fail to exploit any intrinsic low-dimensionality, resulting in a predicted speed-up ratio close to 1.

The expression (15) can be interpreted as a relative speed-up ratio, indicating whether `UPOQA` delivers the expected performance gain. A relative speed-up ratio greater than 1 suggests that revealing the partially separable structure not only allows the algorithm to exploit intrinsic low-dimensionality for reduced modeling cost but also enables the distillation of the primary nonconvexity into relatively low-dimensional elements, further lowering the actual cost. If `UPOQA` fails to converge regardless of whether the partially separable structure is utilized, such cases are excluded when computing $\mathrm{su}(\alpha)$.

To present the results more intuitively, we design $\mathrm{su}(\alpha)$ to be computed across two distinct regimes: $\alpha \geq 1$ and $0 \leq \alpha < 1$. For $\alpha \geq 1$, $\mathrm{su}(\alpha)$ represents the proportion of problems where `UPOQA`, after exploiting the partially separable structure, achieves a relative speed-up ratio $c_p \geq 1$ while maintaining $c_p \leq \alpha$. A higher value in this regime is desirable, and the corresponding curve in the speed-up profile plot should ideally cluster toward the upper-left region. Specifically, $\mathrm{su}(+\infty)$ quantifies the proportion of problems with $c_p \geq 1$. For the regime $0 \leq \alpha < 1$, $\mathrm{su}(\alpha)$ measures the proportion of problems where $c_p < 1$ yet `UPOQA` attains a relative speed-up ratio no less than α, while $\mathrm{su}(0)$ indicates the proportion of problems with $c_p < 1$.

Furthermore, we always expect the plotted speed-up profile curve to exhibit a roughly balanced distribution across $\alpha = 1$, and to rise rapidly on both sides of it. This implies that for the majority of problems, the predicted speed-up ratio $n_p / \max_{i=1,\ldots,q_p} n_{p,i}$ approximates the actual speed-up ratio achieved.

3.2 Test Results on `CUTEst` Problems

3.2.1 Test Problems. Since its release in 1995, the `CUTEst` testing environment and problem set [20] has been widely used by researchers in numerical optimization for designing, testing, and comparing both constrained and unconstrained optimization algorithms. Each test problem is described in the *standard input format* (SIF), which utilizes the group partially separable structure [14] defined as:

$$f(x) = \sum_{i \in \mathcal{G}_{\mathrm{obj}}} \frac{F_i\left(a_i(x), \omega_i\right)}{\sigma_i} + \frac{1}{2} x^\top H x, \tag{16}$$

where each term $F_i\left(a_i(x), \omega_i\right)/\sigma_i$ is called an objective-function group, $H \in \mathbb{R}^{n \times n}$ is symmetric, $\mathcal{G}_{\mathrm{obj}}$ is an index set, and F_i is a linear or nonlinear function of $a_i(x)$ and ω_i. We employ the `S2MPJ` tool [21] as the parser for SIF files to create partially separable versions of these problems with Python based on the group partially separable structure (16). In our implementation, each $F_i\left(a_i(x), \omega_i\right)/\sigma_i$ forms an element, and the quadratic term $x^\top H x/2$ constitutes a separate element, together forming the partially separable structure of a problem.

We conducted a screening of available CUTEst problems, excluding those that are inconvenient for testing or have undesirable structures, including:

- Constrained problems;
- Problems with excessively large average elemental dimension $\overline{n} = \left(\sum_{i=1}^{q} n_i\right)/q$, indicating a lack of intrinsic low-dimensionality;
- Problems with too many ($q \gg n$) or too few ($q = 1$ or 2) elements;
- Problems requiring excessively long construction time;
- Problems that are highly difficult to optimize, where none of the solvers in $\mathcal{S}$ can converge within the given evaluation budget;
- Problems with excessively small or large dimensions.

Subsequently, we selected a test set of medium-scale problems, whose basic information is provided in Table 2 in Appendix A. This set comprises 85 partially separable problems with dimensions $n \in [21,\ 100]$, most of which are 50-dimensional. These problems typically contain a substantial number of elements, with many satisfying $q \geq n$. Whether this structure is sufficiently representative of real-world applications remains debatable. In contrast, the quantum variational problems discussed later satisfy $q \ll n$.

3.2.2 Test Results. We tested the UPOQA, UPOQA (single), BOBYQA, and L-BFGS-B (ffd) algorithms on the 85 problems included in Table 2 in Appendix A. Here, UPOQA (single) denotes optimizing the objective function as a whole element without exploiting the partially separable structure. The BOBYQA [48] implementation was from version 2.8.0 of Python's nlopt library, while L-BFGS-B was from version 1.14.1 of the scipy library, using forward-finite-difference gradients for the objective. For each problem, the maximum number of function evaluations allowed was set to $\max\{1000n, 10000\}$, with the same limit applied to each element function evaluation in UPOQA. For the convergence criterion (14), we selected $\varepsilon = 10^{-1},\ 10^{-3},\ 10^{-5}$, and 10^{-7}, obtaining the performance profiles in Figure 4 and the data profiles shown in Figure 5. Table 1 lists the number of function evaluations required for convergence on the five problems of largest dimensions in the test set, where ∞ indicates failure to converge within the evaluation budget.

Figures 4 and 5, along with Table 1, demonstrate that UPOQA achieves significant acceleration by exploiting the partially separable structure. Across all convergence tolerances, it solves more problems with fewer evaluations. While UPOQA (single) solves slightly fewer problems than BOBYQA and L-BFGS-B (ffd), this is understandable since it was not specifically designed for single-objective optimization. For all tolerances, UPOQA solves at least 76.4% of problems with fewest evaluations, reaching 90.6% at $\varepsilon = 10^{-1}$, indicating superior efficiency for low-accuracy solutions. Remarkably, UPOQA's success rate declines only marginally as ε decreases. At $\varepsilon = 10^{-7}$, it solves approximately 5% more problems than the next best performer L-BFGS-B (ffd), while their performance was nearly identical at $\varepsilon = 10^{-1}$. This demonstrates UPOQA's robustness in high-accuracy scenarios.

To further evaluate the acceleration effect of UPOQA, we present the speed-up profiles computed by UPOQA and UPOQA (single) in Figure 6. At a low convergence tolerance ($\varepsilon = 10^{-1}$), the profile shows that among problems where at least one algorithm converged, 73.5% exhibited relative speed-up ratios between 0.5 and 2 (i.e., the gray-shaded region $[-1, 1]$ on the horizontal axis), indicating reasonable alignment between predictions and empirical results. At the same time, UPOQA achieved lower speed-up than predicted on 57.6% of problems, outperformed predictions on 35.2%, while failing to converge within the budget on the remaining 7.2%—regardless of exploiting partially separable structures. Thus, UPOQA's acceleration advantage is limited when only low-precision solutions are required. However, at higher precision ($\varepsilon = 10^{-4}$), its acceleration becomes significantly enhanced, with a notably increased proportion of

Table 1. Function evaluations required for convergence by UPOQA, UPOQA (single), BOBYQA, and L-BFGS-B (ffd) on the five largest-dimensional problems from the test set in Table 2 in Appendix A, with convergence tolerances $\varepsilon = 10^{-1},\ 10^{-3},\ 10^{-5}$, and 10^{-7}. For UPOQA, the reported evaluations are the maximum evaluations across all elements. Here, n denotes the problem dimension, q denotes the number of elements in its partially separable structure, $n_1, \ldots, n_q$ are dimensionalities of elements, and ∞ indicates failure to converge within the evaluation budget.

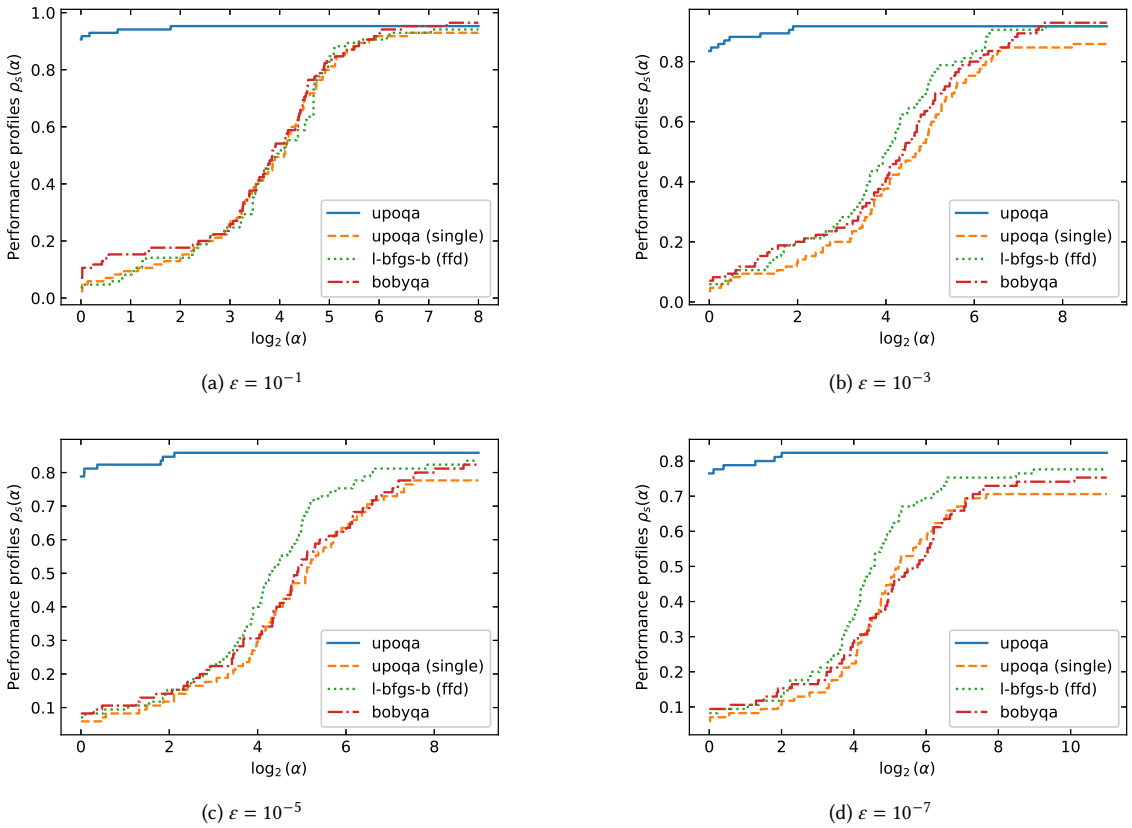
Problem	n	q	$\max n_i$	$\sum n_i/q$	$\log_{10}\varepsilon$	Evaluations			
						UPOQA	UPOQA (single)	BOBYQA	L-BFGS-B (ffd)
HIMMELBI	100	20	5	5.00	-1	**48**	68	57	113
					-3	83	97	**72**	113
					-5	885	2357	∞	**841**
					-7	**20237**	∞	∞	∞
JANNSON3	100	102	2	1.01	-1	**12**	113	116	154
					-3	**47**	429	976	358
					-5	**83**	1309	2349	562
					-7	**108**	1850	3640	1174
LUKSAN21LS	100	100	3	2.98	-1	**11**	246	115	103
					-3	**27**	1033	588	307
					-5	**39**	1318	838	511
					-7	**51**	1517	1098	715
LINVERSE	99	147	4	2.98	-1	**51**	488	489	562
					-3	**139**	1607	1815	1735
					-5	**145**	2880	2486	2551
					-7	**152**	3386	3712	3622
HYDC20LS	99	99	14	7.47	-1	**11**	101	101	52
					-3	**19**	101	101	52
					-5	**19**	101	101	256
					-7	**27**	101	101	358

problems distributed on the $\log_2(\alpha) \geq 0$ side. When tolerance tightens further to $\varepsilon = 10^{-7}$, the outperformance ratio shows no significant change, but the proportion of problems with $\log_2(\alpha) < 0$ decreases markedly—a consequence of reduced solve success rates under stricter precision demands. Overall, the acceleration benefits of UPOQA are marginal in low-precision scenarios but become more pronounced in high-precision contexts.

3.3 Test Results on Quantum Variational Problems

With the advancement of quantum technology, the *variational quantum algorithm* (VQA) has garnered increasing attention as a hybrid quantum-classical approach for solving optimization problems [8]. In brief, given an objective function, the first step of VQA involves constructing a parameterized quantum circuit on a quantum computer to represent this function. Classical derivative-free optimization algorithms are then employed to adjust these parameters and ultimately find the minimum of the objective. Now, let us consider a quantum variational problem:

$$\min_{\theta_1,\ldots,\theta_p \,\in\, \mathbb{R}^{n_\theta}} \quad \sum_{i=1}^{p} w_i \left\langle \phi_i \middle| U(\theta_i)^\dagger \widehat{H} U(\theta_i) \middle| \phi_i \right\rangle + \lambda \sum_{i<j} \left| \langle \phi_i \middle| U(\theta_i)^\dagger U(\theta_j) \middle| \phi_j \rangle \right|^2, \tag{17}$$



(a) $\varepsilon = 10^{-1}$ (b) $\varepsilon = 10^{-3}$

(c) $\varepsilon = 10^{-5}$ (d) $\varepsilon = 10^{-7}$

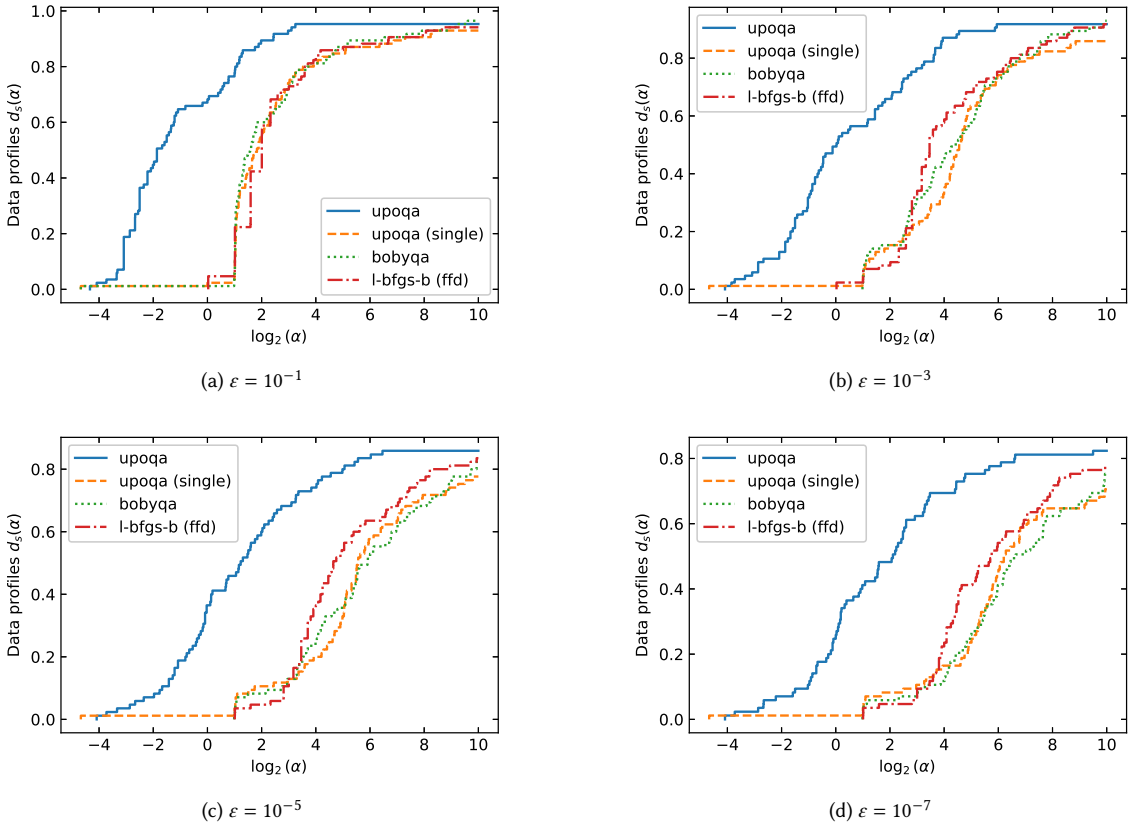
Fig. 4. Performance profiles of UPOQA, UPOQA (single), BOBYQA, and L-BFGS-B (ffd) on the test problems from Table 2 in Appendix A, with convergence tolerances $\varepsilon = 10^{-1}$, 10^{-3}, 10^{-5}, and 10^{-7}.

where $\widehat{H}$ denotes the Hamiltonian operator of the molecular system, $U(\theta_1), \ldots, U(\theta_p)$ represent the parameterized ansatz quantum circuits with parameters $\theta_1, \ldots, \theta_p$ respectively, $\phi_1, \ldots, \phi_p$ are the initial quantum states, $w_1, \ldots, w_p > 0$ are the weights, and $\lambda > 0$ is the penalty coefficient. The goal of this problem is to determine the states corresponding to the smallest p energy levels of a given molecular system.

In equation (17), each term under the summation represents the measurement outcome of a quantum circuit, which is typically quite costly. We treat each term in (17) as a black box and reformulate the problem as:

$$\min_{\theta_1, \ldots, \theta_p \in \mathbb{R}^{n_\theta}} \sum_{i=1}^{p} f_i(\theta_i) + \sum_{1 \le i < j \le p} f_{ij}(\theta_i, \theta_j), \tag{18}$$

where $f_i(\theta_i) = w_i \left\langle \phi_i \left| U(\theta_i)^\dagger \widehat{H} U(\theta_i) \right| \phi_i \right\rangle$, $f_{ij}(\theta_i, \theta_j) = \lambda \left| \left\langle \phi_i \left| U(\theta_i)^\dagger U(\theta_j) \right| \phi_j \right\rangle \right|^2$. Clearly, this is a coordinate partially separable problem, where the objective function is np-dimensional and consists of $p(p+1)/2$ elements with a maximum dimensionality of $2n$ and exhibits a dense Hessian matrix. Similar structures are common in other quantum variational problems [3, 4].



(a) $\varepsilon = 10^{-1}$ (b) $\varepsilon = 10^{-3}$

(c) $\varepsilon = 10^{-5}$ (d) $\varepsilon = 10^{-7}$

Fig. 5. Data profiles of UPOQA, UPOQA `(single)`, BOBYQA, and L-BFGS-B `(ffd)` on the test problems from Table 2 in Appendix A, with convergence tolerances $\varepsilon = 10^{-1}$, 10^{-3}, 10^{-5}, and 10^{-7}.

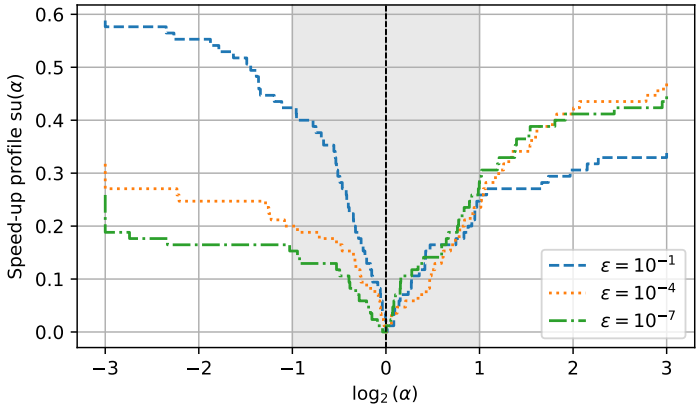
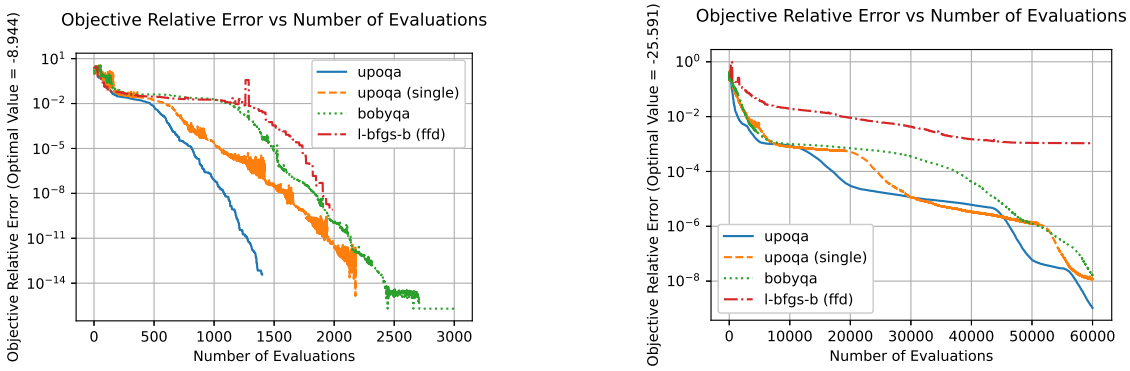


Fig. 6. Speed-up profiles computed by UPOQA and UPOQA `(single)` on the test problems from Table 2 in Appendix A, with convergence tolerances $\varepsilon = 10^{-1}$, 10^{-4}, and 10^{-7}.



(a) H_2 molecule

(b) H_4 molecule

Fig. 7. Objective function value descent curves when optimizing the noiseless quantum variational problem (18) for H_2 and H_4 molecular systems using UPOQA, UPOQA (single), BOBYQA, and L-BFGS-B (ffd).

We conducts numerical experiments on problem (18) for H_2 and a model consisting of four hydrogen atoms arranged in a square lattice, which is denoted as H_4, in the STO-3G basis with UCCSD [52] ansatz circuits. The UCCSD block pattern is repeated for three times in order to increase the expressiveness of the ansatz. We set $p = 3$, indicating that the goal is to compute the three lowest-energy excited states and their state energies, with weights set as $w_i = i$ for $i = 1, 2, 3$. The set of initial states ϕ_1, ϕ_2, ϕ_3 and all molecular Hamiltonians are chosen to be the Hartree-Fock state and low-lying single-particle excitations above it. The starting points $\theta_{0,1}, \theta_{0,2}, \theta_{0,3}$ are randomly sampled according to a uniform distribution on $[-2\pi, 2\pi)$. Both experiments are performed in a noiseless environment. The experimental code is implemented in Python using `qiskit` version 0.37.0, with quantum circuit designs consistent with those in the quantum orbital minimization method (qOMM) [3]. In the H_2 experiment, the interatomic distance is set to 0.735 Å, and the penalty coefficient is empirically chosen as $\lambda = 12$. For the H_4 experiment, the interatomic distance is 1.23 Å, with $\lambda = 30$.

The experimental results from both tests are shown in Fig 7. UPOQA demonstrates superior performance on both problems, with more pronounced acceleration effects in the H_2 experiment. Since the predicted acceleration rate is $p/2 = 1.5$, the results align well with theoretical predictions. In both experiments, all solvers encountered "plateau periods" where the objective function value stagnated, but UPOQA consistently escaped these phases faster and achieved solutions with fewer function evaluations. These results indicate that leveraging the internal structure of problem (18) indeed leads to acceleration, and suggest that UPOQA could deliver outstanding performance on larger-scale problems and other general quantum variational problems.

4 CONCLUSION

We propose the UPOQA algorithm for derivative-free optimization of partially separable problems. Based on quadratic interpolation models and a trust-region framework, UPOQA constructs underdetermined quadratic models for each element function, employing the Steinmetz projection and a modified projected gradient method to solve structured trust-region subproblems. The implementation incorporates Powell's techniques [45, 47], resulting in low iteration costs and strong robustness. The trust-region radius management strategy is modified from Shahabuddin's criterion [55,

Section 2.4]. Additionally, UPOQA features practical functionalities such as starting point search, restart mechanism, and hybrid black-white-box optimization. Comprehensive numerical experiments on CUTEst problems and a set of quantum variational problems validate the algorithm's effectiveness and robustness in problem-solving.

We acknowledge several aspects of UPOQA that warrant further research and improvement. The modified projected gradient method currently exhibits limited single-step descent rates and requires excessive iterations, leading to significant runtime overhead in some scenarios. Theoretical analysis remains incomplete, particularly regarding convergence rates and their relationship with partially separable structures. Experimentally, we aspire to test the algorithm on more real-world applications, as the artificially constructed partially separable structures from CUTEst test problems may lack sufficient representativeness. Future enhancements could potentially incorporate state-of-the-art DFO techniques such as sample averaging and regression models [7], as well as preconditioning and subspace methods [59], which might further improve UPOQA's performance.

REFERENCES

[1] Albert S. Berahas, Liyuan Cao, Krzysztof Choromanski, and Katya Scheinberg. 2022. A theoretical and empirical comparison of gradient approximations in derivative-free optimization. *Found. Comut. Math.* 22, 2 (2022), 507–560.

[2] Dimitri P. Bertsekas et al. 2011. Incremental gradient, subgradient, and proximal methods for convex optimization: A survey. *Optimization for Machine Learning* 2010, 1-38 (2011), 3.

[3] Joel Bierman, Yingzhou Li, and Jianfeng Lu. 2022. Quantum orbital minimization method for excited states calculation on a quantum computer. *J. Chem. Theory Comput.* 18, 8 (2022), 4674–4689.

[4] Joel Bierman, Yingzhou Li, and Jianfeng Lu. 2023. Improving the Accuracy of Variational Quantum Eigensolvers with Fewer Qubits Using Orbital Optimization. *J. Chem. Theory Comput.* 19, 3 (2023), 790–798.

[5] Zyed Bouzarkouna, Anne Auger, and Didier Y. Ding. 2011. Local-meta-model CMA-ES for partially separable functions. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* (Dublin, Ireland) *(GECCO '11)*. Association for Computing Machinery, New York, NY, USA, 869–876.

[6] Huiping Cao and Lan Yao. 2016. A partitioned PSB method for partially separable unconstrained optimization problems. *Appl. Math. Comput.* 290 (2016), 164–177.

[7] Coralia Cartis, Jan Fiala, Benjamin Marteau, and Lindon Roberts. 2019. Improving the flexibility and robustness of model-based derivative-free optimization solvers. *ACM Trans. Math. Softw.* 45, 3 (2019), 1–41.

[8] Marco Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C. Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R. McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, et al. 2021. Variational quantum algorithms. *Nat. Rev. Phys.* 3, 9 (2021), 625–644.

[9] Benoît Colson and Philippe L. Toint. 2001. Exploiting band structure in unconstrained optimization without derivatives. *Optim. Eng.* 2 (2001), 399–412.

[10] Benoît Colson and Philippe L. Toint. 2002. A derivative-free algorithm for sparse unconstrained optimization problems. In *Trends in Industrial and Applied Mathematics: Proceedings of the 1st International Conference on Industrial and Applied Mathematics of the Indian Subcontinent*. Springer US, Boston, MA, 131–147.

[11] Benoît Colson and Philippe L. Toint. 2005. Optimizing partially separable functions without derivatives. *Optim. Methods Softw.* 20, 4-5 (2005), 493–508.

[12] Andrew R. Conn, Nicholas I. M. Gould, Annick Sartenaer, and Philippe L. Toint. 1996. Convergence properties of minimization algorithms for convex constraints using a structured trust region. *SIAM J. Optim.* 6, 4 (1996), 1059–1086.

[13] Andrew R. Conn, Nicholas I. M. Gould, and Philippe L. Toint. 1994. *Improving the Decomposition of Partially Separable Functions in the Context of Large-Scale Optimization: a First Approach.* Springer US, Boston, MA, 82–94.

[14] Andrew R. Conn, Nicholas I. M. Gould, and Philippe L. Toint. 2010. *LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization (Release A)* (1st ed.). Vol. 17. Springer Publishing Company, Incorporated.

[15] Andrew R. Conn, Katya Scheinberg, and Luis N. Vicente. 2009. *Introduction to Derivative-Free Optimization.* SIAM, USA.

[16] Andrew R. Conn and Philippe L. Toint. 1996. *An Algorithm using Quadratic Interpolation for Unconstrained Derivative Free Optimization.* Springer US, Boston, MA, 27–47.

[17] Carlos H. da Silva Santos, Marcos S. Goncalves, and Hugo E. Hernandez-Figueroa. 2010. Designing novel photonic devices by bio-inspired computing. *IEEE Photonics Technol. Lett.* 22, 15 (2010), 1177–1179.

[18] Michel J. Daydé, Jean-Yves L'Excellent, and Nicholas I. M. Gould. 1997. Element-by-element preconditioners for large partially separable optimization problems. *SIAM J. Sci. Comput.* 18, 6 (1997), 1767–1787.

[19] Daisy Y. Ding and Christopher F. McKee. 2011. Using partial separability of the objective function for gradient-based optimizations in History Matching. In *SPE Reservoir Simulation Conference*, Vol. SPE Reservoir Simulation Symposium. SPE, The Woodlands, Texas, USA, SPE–140811–MS.

[20] Nicholas I. M. Gould, Dominique Orban, and Philippe L. Toint. 2015. CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. *Comput. Optim. Appl.* 60 (2015), 545–557.

[21] Serge Gratton and Philippe L. Toint. 2024. S2MPJ and CUTEst optimization problems for Matlab, Python and Julia. arXiv:2407.07812 [math.OC]

[22] Andreas Griewank and Philippe L. Toint. 1982. *On the Unconstrained Optimization of Partially Separable Functions.* Academic Press, London, 301–312.

[23] Andreas Griewank and Philippe L. Toint. 1982. Partitioned variable metric updates for large structured optimization problems. *Numer. Math.* 39, 1 (1982), 119–137.

[24] Andreas Griewank and Philippe L. Toint. 1984. On the existence of convex decompositions of partially separable functions. *Math. Program.* 28 (1984), 25–49.

[25] James C. Gross and Geoffrey T. Parks. 2022. Optimization by moving ridge functions: derivative-free optimization for computationally intensive functions. *Eng. Optim.* 54, 4 (2022), 553–575.

[26] Nikolaus Hansen. 2006. *The CMA Evolution Strategy: A Comparing Review.* Springer Berlin Heidelberg, Berlin, Heidelberg, 75–102.

[27] Willi Hock and Klaus Schittkowski. 1980. Test examples for nonlinear programming codes. *J. Optim. Theory Appl.* 30 (1980), 127–129.

[28] Matthew Hough and Lindon Roberts. 2022. Model-based derivative-free methods for convex-constrained optimization. *SIAM J. Optim.* 32, 4 (2022), 2552–2579.

[29] Bulent Karasözen. 2007. Survey of derivative free optimization methods based on interpolation. *J. Ind. Manag. Optim.* 3, 2 (2007), 321–334.

[30] Kamer Kaya, Figen Öztoprak, Ş. İlker Birbil, A. Taylan Cemgil, Umut Şimşekli, Nurdan Kuru, Hazal Koptagel, and M. Kaan Öztürk. 2019. A framework for parallel second order incremental optimization algorithms for solving partially separable problems. *Comput. Optim. Appl.* 72 (2019), 675–705.

[31] Stefan Kern, Nikolaus Hansen, and Petros Koumoutsakos. 2006. Local meta-models for optimization using evolution strategies. In *Parallel Problem Solving from Nature - PPSN IX.* Springer Berlin Heidelberg, Berlin, Heidelberg, 939–948.

[32] Scott Kirkpatrick, Charles D. Gelatt Jr., and Mario P. Vecchi. 1983. Optimization by simulated annealing. *Science* 220, 4598 (1983), 671–680.

[33] M. Lescrenier. 1988. Partially separable optimization and parallel computing. *Ann. Oper. Res.* 14 (1988), 213–224.

[34] Adrian Lewis, Russell Luke, and Jerome Malick. 2007. Local convergence for alternating and averaged nonconvex projections. arXiv:0709.0109 [math.OC]

[35] Robert M. Lewis and Virginia Torczon. 1999. Pattern search algorithms for bound constrained minimization. *SIAM J. Optim.* 9, 4 (1999), 1082–1099.

[36] Robert M. Lewis, Virginia Torczon, and Michael W. Trosset. 2000. Direct search methods: then and now. *J. Comput. Appl. Math.* 124, 1-2 (2000), 191–207.

[37] Xiaodong Li, Ke Tang, Mohammad N. Omidvar, Zhenyu Yang, and Kai Qin. 2013. Benchmark functions for the CEC 2013 special session and competition on large-scale global optimization. *Gene* 7, 33 (2013), 8.

[38] Jorge J. Moré and Stefan M. Wild. 2009. Benchmarking derivative-free optimization algorithms. *SIAM J. Optim.* 20, 1 (2009), 172–191.

[39] Rodrigue Oeuvray. 2005. *Trust-Region Methods Based on Radial Basis Functions with Application to Biomedical Imaging.* Ph. D. Dissertation. École polytechnique fédérale de Lausanne, Lausanne.

[40] Margherita Porcelli and Philippe L. Toint. 2022. Exploiting problem structure in derivative free optimization. *ACM Trans. Math. Softw.* 48, 1 (2022), 1–25.

[41] Michael J. D. Powell. 1964. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Comput. J.* 7, 2 (1964), 155–162.

[42] Michael J. D. Powell. 1965. A method for minimizing a sum of squares of non-linear functions without calculating derivatives. *Comput. J.* 7, 4 (1965), 303–307.

[43] Michael J. D. Powell. 1994. *A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation.* Springer Netherlands, Dordrecht. 51–67 pages.

[44] Michael J. D. Powell. 2004. Least Frobenius norm updating of quadratic models that satisfy interpolation conditions. *Math. Program.* 100, 1 (May 2004), 183–215.

[45] Michael J. D. Powell. 2004. On updating the inverse of a KKT matrix. *Numerical Linear Algebra and Optimization (Science Press, Beijing)* (2004), 56–78.

[46] Michael J. D. Powell. 2006. *The NEWUOA Software for Unconstrained Optimization without Derivatives.* Springer US, Boston, MA, 255–297.

[47] Michael J. D. Powell. 2008. Developments of NEWUOA for minimization without derivatives. *IMA J. Numer. Anal.* 28, 4 (02 2008), 649–664.

[48] Michael J. D. Powell. 2009. The BOBYQA algorithm for bound constrained optimization without derivatives. *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge* 26 (2009), 26–46.

[49] Christopher J. Price and Philippe L. Toint. 2006. Exploiting problem structure in pattern search methods for unconstrained optimization. *Optim. Methods Softw.* 21, 3 (2006), 479–491.

[50] Tom M. Ragonneau. 2023. *Model-based derivative-free optimization methods and software.* Ph. D. Dissertation. Hong Kong Polytechnic University, Hong Kong, China.

[51] Tom M. Ragonneau and Zaikun Zhang. 2024. PDFO: a cross-platform package for Powell's derivative-free optimization solvers. *Math. Program. Comput.* 16, 4 (2024), 535–559.

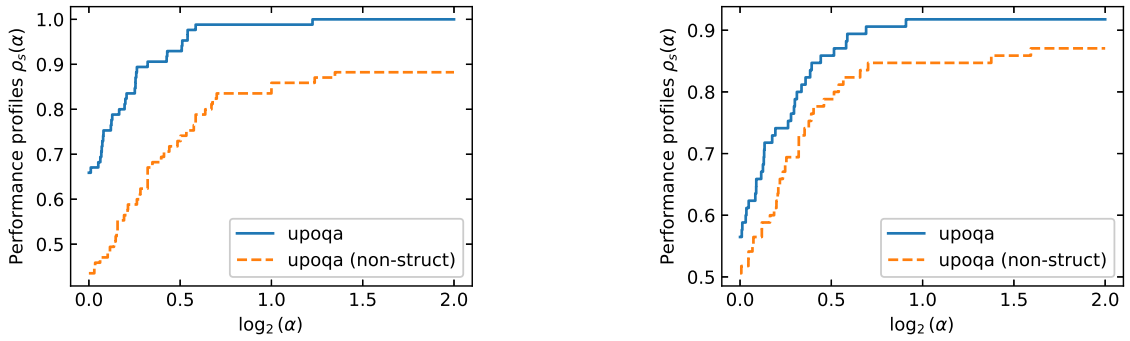
[52] Jonathan Romero, Ryan Babbush, Jarrod R. McClean, Cornelius Hempel, Peter J. Love, and Alán Aspuru-Guzik. 2018. Strategies for quantum computing molecular energies using the unitary coupled cluster ansatz. *Quantum Sci. Technol.* 4, 1 (2018), 014008.
[53] Raymond Ros and Nikolaus Hansen. 2008. A simple modification in CMA-ES achieving linear time and space complexity. In *International Conference on Parallel Problem Solving from Nature*. Springer, Springer Berlin Heidelberg, Berlin, Heidelberg, 296–305.
[54] Thomas P. Runarsson and Xin Yao. 2005. Search biases in constrained evolutionary optimization. *IEEE Trans. Syst. Man. Cybern. Pt. C. Appl. Rev.* 35, 2 (2005), 233–243.
[55] Johara S. Shahabuddin. 1996. *Structured Trust Region Algorithms for the Minimization of Nonlinear Functions*. Ph. D. Dissertation. Cornell University, USA.
[56] Philippe L. Toint. 1983. *Test Problems for Partially Separable Optimization and Results for the Routine PSPMIN*. Technical Report. The University of Namur, Department of Mathematics.
[57] Philippe L. Toint. 1986. Global convergence of the partitioned BFGS algorithm for convex partially separable optimization. *Math. Program.* 36 (1986), 290–306.
[58] Hongchao Zhang and Andrew R. Conn. 2012. On the local convergence of a derivative-free algorithm for least-squares minimization. *Comput. Optim. Appl.* 51, 2 (2012), 481–507.
[59] Zaikun Zhang. 2025. Scalable derivative-free optimization algorithms with low-dimensional subspace techniques. arXiv:2501.04536 [math.OC]
[60] Yu Zhu, Li Zhang, Rushi Lan, and Xiaonan Luo. 2019. Large-scale partially separable function optimization using cooperative coevolution and competition strategies. In *2019 Eleventh International Conference on Advanced Computational Intelligence (ICACI)*. IEEE, Guilin, China, 144–148.

A TEST PROBLEMS

Table 2 lists all the test problems used in Section 3.2. These problems were extracted from the `CUTEst` problem set [20] using the `S2MPJ` tool [21].

B PERFORMANCE OF `UPOQA` USING UNSTRUCTURED TRUST REGIONS

We also compared the performance of `UPOQA` with and without structured trust regions. When structured trust regions are not employed, all the radii $\Delta_{k,1}, \ldots, \Delta_{k,q}$ remain identical, and the trust-region subproblem (6) reduces to an optimization problem within a spherical region. We solve it using the truncated conjugate gradient method with a boundary improvement step, as implemented in `COBYQA` [50]. This variant of `UPOQA` is labeled `UPOQA (non-struct)`.



(a) $\varepsilon = 10^{-3}$ (b) $\varepsilon = 10^{-7}$

Fig. 8. Performance profiles of `UPOQA` and its unstructured-trust-region variant `UPOQA (non-struct)` on the test problems from Table 2 in Appendix A, with convergence tolerances $\varepsilon = 10^{-3}$ and 10^{-7}.

We plot the performance profiles for both algorithms under convergence tolerances of $\varepsilon = 10^{-3}$ and 10^{-7}, as shown in Figure 8. At $\varepsilon = 10^{-3}$, `UPOQA` significantly outperforms `UPOQA (non-struct)`. With structured trust regions, `UPOQA`

Table 2. CUTEst test problems used in Section 3.2, where n denotes the problem dimension, q denotes the number of elements, and $n_1, \ldots, n_q$ are the dimensions of elements.

Problem	n	q	$\max n_i$	$\sum n_i/q$	Problem	n	q	$\max n_i$	$\sum n_i/q$
ANTWERP	27	19	19	7.74	ARWHEAD	50	98	2	1.50
BDQRTIC	50	92	5	3.00	BIGGSB1	50	51	2	1.96
BROYDN3DLS	50	50	3	2.96	BROYDNBDLS	50	50	7	6.68
CHNROSNB	50	98	2	1.50	CHNRSNBM	50	98	2	1.50
COSINE	50	49	2	2.00	CURLY10	50	50	11	9.90
CURLY20	50	50	21	16.80	CURLY30	50	50	31	21.70
CVXQP1	50	50	3	2.94	CYCLOOCFLS	86	60	6	5.73
CYCLOOCTLS	90	60	6	6.00	DIXON3DQ	50	50	2	1.96
DQRTIC	50	50	1	1.00	DTOC2	88	15	6	5.87
EG2	50	50	2	1.96	ENGVAL1	50	98	2	1.50
ERRINROS	50	98	2	1.50	ERRINRSM	50	98	2	1.50
EXTROSNB	50	50	2	1.98	FLETBV3M	50	102	50	1.96
FLETCBV2	50	151	2	1.32	FLETCHCR	50	98	2	1.50
FREUROTH	50	98	2	2.00	GILBERT	50	50	1	1.00
HATFLDC	25	25	2	1.92	HIMMELBI	100	20	5	5.00
HUESmMOD	50	50	1	1.00	HYDC20LS	99	99	14	7.47
HYDCAR6LS	29	29	14	6.69	INDEFM	50	98	3	1.98
JANNSON3	100	102	2	1.01	JANNSON4	50	52	2	1.02
LEVYMONT10	50	100	2	1.49	LEVYMONT6	50	100	2	1.49
LIARWHD	50	100	2	1.49	LINVERSE	99	147	4	2.98
LUKSAN21LS	100	100	3	2.98	LUKVLI10	50	50	2	2.00
LUKVLI11	50	64	2	1.25	LUKVLI13	50	48	2	1.67
LUKVLI14	50	64	2	1.25	LUKVLI1	50	98	2	1.50
LUKVLI2	50	125	2	1.62	LUKVLI3	50	96	2	2.00
LUKVLI4C	50	120	2	1.60	LUKVLI5	52	50	3	3.00
LUKVLI8	50	40	5	4.00	LUKVLI9	50	51	50	2.45
METHANB8LS	31	31	11	6.29	METHANL8LS	31	31	11	6.29
MOREBV	50	50	3	2.96	NCB20B	50	50	20	12.78
NCB20	60	51	30	12.75	NONDQUAR	50	50	3	2.96
NONSCOMP	50	50	2	1.98	OSCIGRAD	50	50	3	2.96
OSCIPATH	50	50	2	1.98	PENALTY1	50	51	50	1.96
PENALTY2	50	100	50	1.98	QING	50	50	1	1.00
RAYBENDL	62	30	4	4.00	SANTALS	21	23	4	3.52
SBRYBND	50	50	7	6.68	SCHMVETT	50	48	3	3.00
SCOSINE	50	49	2	2.00	SINEALI	50	50	2	1.98
SINQUAD2	50	50	3	2.94	SINROSNB	50	50	2	1.98
SPARSINE	50	50	6	5.58	SPARSQUR	50	50	6	5.58
SSBRYBND	50	50	7	6.68	STRTCHDV	50	49	2	2.00
TOINTGOR	50	83	5	1.81	TOINTGSS	50	48	3	3.00
TOINTPSP	50	83	5	1.81	TOINTQOR	50	83	5	1.81
TQUARTIC	50	50	2	1.98	TRIDIA	50	50	2	1.98
VARDIM	50	52	50	2.88	VAREIGVL	51	51	50	13.88
YAO	52	52	1	1.00					

achieves faster convergence on 65.8% of the problems, whereas UPOQA (non-struct) exhausts its evaluation budget without reaching the same solution accuracy as UPOQA on 11.7% of the problems. At $\varepsilon = 10^{-7}$, the performance gap narrows slightly, with UPOQA converging first on 56.4% of the problems. These results demonstrate the effectiveness of replacing the unstructured trust region with a structured one.