Mathematical Programming Computation (2024) 16:665–693
https://doi.org/10.1007/s12532-024-00267-7

FULL LENGTH PAPER

Solving higher dimensional expensive black box global optimization problems using sparse directional search on surrogates

Limeng Liu[1] · Muming Yang[2] · Christine A. Shoemaker[1] · Tingting Xie[3]

Received: 8 September 2022 / Accepted: 12 August 2024 / Published online: 16 October 2024
© The Author(s) 2024

Abstract
We present a new algorithm Directional Optimization Search with Surrogate (DOSS), for optimizing problems with box constraints and a computationally expensive black-box objective function. DOSS is a radial basis function (RBF) based method that mainly focuses on higher dimensional and computationally expensive objective functions that can be multimodal. DOSS introduces three new techniques not previously used in earlier RBF algorithms, including using a combination of the coordinates knowledge level, fewer initial sampling points, and the surrogate's gradient information. Numerical results on a test set including 14 test problems with 36, 48, and 60 dimensions show that DOSS outperforms two recently published algorithms RBFOpt and TuRBO, and earlier RBF algorithms such as DYCORS. TuRBO is a Gaussian process based optimization algorithm, which outperformed earlier state-of-the-art methods. DOSS algorithm also has a good performance on real-world optimization problems, including robot pushing and rover trajectory planning problems. Almost sure convergence for the DOSS algorithm is also proven in this paper. An implementation of DOSS is available at https://doi.org/10.5281/zenodo.13731558.

Mathematics Subject Classification 90-08 Computational methods for problems pertaining to operations research and mathematical programming · 90C26 Nonconvex programming, global optimization

✉ Tingting Xie
tingtingxie@u.nus.edu

[1] Department of Industrial Systems Engineering and Management, National University of Singapore, Singapore, Singapore

[2] Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China

[3] Institute of High Performance Computing (IHPC), Agency for Science, Technology and Research (A*STAR), Singapore, Singapore

1 Introduction

Higher dimensional, computationally expensive, and black-box optimization problems (HEB) for which no derivative information exists are an important class of problems, which are involved in many real-world optimization problems. However, the optimization of HEB problems is extremely hard. On the one hand, optimization difficulties grow rapidly with dimensions. On the other hand, mathematical expressions, including derivatives, are unavailable in black-box problems. Moreover, the computationally expensive problems restrict us from solving these problems within a limited number of evaluations. The goal of this paper is to develop an efficient algorithm for HEB problems.

1.1 Related work

Given the complexity and diversity of HEB problems, a wide array of optimization algorithms have been developed to solve them. Among these, surrogate-based optimization algorithms stand out for their ability to build and utilize an approximation (a surrogate) of the true objective function based on objective function evaluations made in previous steps of the algorithm. This surrogate provides a computationally cheaper means to explore the solution space, making them invaluable for identifying promising new evaluation points across the problem's domain.

Surrogate-based optimization strategies primarily fall into two categories. The first category leverages the Gaussian process (GP) model, often referred to as Bayesian optimization (BO), while the second employs the radial basis function (RBF) model. Early work by [14] on the GP model introduced the concept of Expected Improvement (EI) for point selection, laying the groundwork for numerous studies that have since advanced BO techniques for various applications [2, 3, 5, 9, 15, 23, 25, 26]. While GP models, including the Efficient Global Optimization (EGO) approach, are traditionally favored for problems of modest dimensionality (typically fewer than 11 dimensions), notable exceptions exist. The Trust Region Bayesian Optimization (TuRBO, [8]) algorithm exemplifies how trust-region methodologies, which are usually associated with local optimization, can be ingeniously adapted for global optimization challenges.

Similarly, the RBF model has proven to be effective in solving in low-dimensional problems [21]. Subsequent adaptations, like the DYnamic COordinate search using Response Surface models (DYCORS) algorithm, have enhanced its applicability to higher-dimensional spaces. [11] developed an optimization algorithm with RBF and a bumpiness function to obtain the next iterate.

1.2 Main contribution

DOSS builds upon and significantly improves the earlier DYCORS algorithm [22] by adding three novel techniques. They are (1) a new initialization strategy, (2) perturbing coordinates by coordinate knowledge strategy, and (3) a surrogate-based sparse direction search strategy. The proposed DOSS algorithm with all these techniques

outperforms two recently published algorithms, RBFOpt and TuRBO, as well as the original DYCORS method.

2 Problem description

The optimization problem considered in this paper is a box-constrained black-box optimization problem with a single objective and continuous variables. This kind of problem can be formulated in the following form

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & x \in \mathcal{D} \end{aligned}$$

where x is the vector of decision variables, whose domain $\mathcal{D} = [l_i, r_i]^d$ is a closed hyper-rectangle in $\mathbb{R}^d$. l_i and r_i denote the lower bound and upper bound of coordinate i, respectively. The objective function f is continuous and a black box, which means the value of f at a point x can only be obtained by an expensive function evaluation, and the derivative of f is not available. Moreover, f has two important features: it is possibly multi-modal (i.e., it has multiple local minima), and it is very expensive to evaluate, so we only have a limited budget for objective function evaluations to do the optimization.

3 General RBF-based optimization algorithm

Our proposed DOSS algorithm is constructed on the common architecture underlying most RBF-based optimization methods such as DYCORS ([22]) and SRBF ([21]). Thus, prior to detailing the special features of the DOSS algorithm, we use this section to elucidate this general RBF-based algorithmic architecture as well as some features of DYCORS to facilitate the comparison between DYCORS and DOSS in a later section.

3.1 General framework

An RBF-based algorithm consists of three main steps: *initialization*, *iterative loop*, and *termination*. The *initialization* step generates n_0 initial points $x_1, \ldots, x_{n_0}$ by some experimental design, like Latin Hypercube Design. After the evaluation of these points, the initial surrogate model is fitted by $(x_i, f(x_i)), i = 1, \ldots, n_0$. This model will guide the search in the *iterative loop* step.

In the *iterative loop* step, there are four sub-steps: (1) Generate candidate points; (2) Select a new point from candidates; (3) Objective function evaluation; (4) Update surrogate model and parameters.

The *iterative loop* is repeated until the maximum number of function evaluations $N_{\max}$ is reached. Then the algorithm will return the best solutions found so far.

3.2 Surrogate model: radial basis function

The RBF function has the following form

$$s_n(x) = \sum_{i=1}^{n} \lambda_i \phi\left(\|x - x_i\|\right) + g(x), \quad x \in \mathbb{R}^d, \tag{1}$$

where $\|\cdot\|$ is the Euclidean norm, and x_i for $i = 1, \ldots, n$, are previously evaluated points. g is a linear polynomial in d variables. ϕ is the kernel function, which is commonly selected as the cubic form: $\phi(r) = r^3$ because of its successful applications [4, 20]. Coefficients λ and c, the latter being a coefficient within g, can be obtained by solving the following linear system

$$\begin{pmatrix} \Phi & P \\ P^\top & 0 \end{pmatrix} \begin{pmatrix} \lambda \\ c \end{pmatrix} = \begin{pmatrix} F \\ 0 \end{pmatrix}, \tag{2}$$

where

$$P = \begin{pmatrix} (x_1)^T & 1 \\ \vdots & \vdots \\ (x_n)^T & 1 \end{pmatrix}, \lambda = \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{pmatrix}, c = \begin{pmatrix} c_1 \\ \vdots \\ c_{d+1} \end{pmatrix}, F = \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix},$$

and $\Phi = \left(\phi\left(\left\|x_i - x_j\right\|\right)\right)_{1 \le i,j \le n}$. In each iteration, all evaluated points are used to solve linear system (2). More specifically, an incremental strategy is adopted, where an existing LU factorization is updated by computing a Cholesky factorization of the Schur complement system. If this process encounters ill-conditioning issues, a fresh LU factorization will be computed. Linear system (2) has a unique solution if and only if $\text{rank}(P) = d + 1$ ([19]).

An RBF-based algorithm uses the radial basis surrogate function to guide the search. The surrogate model $s_n(x)$ is fitted by all the previously evaluated points $x_1, \ldots, x_n$ so that it can be regarded as a prediction of $f(x)$.

3.3 Optimizing surrogate using metric stochastic RBF method

Previous RBF-based algorithms use stochastic search and local metric strategies to optimize the surrogate model. In each iteration in the *iterative loop* step, the way to optimize the surrogate is to generate many candidate points randomly and pick the one with the best score. The candidate points are generated by adding random perturbations on the current best point x_{best}. Since the surrogate models of higher dimensional problems are usually complex, DYCORS ([22]) combines the dynamic coordinate search strategy, which perturbs only a few coordinates on x_{best} to increase the chance of preserving the values of the coordinate decision variable that are already good in some of the dimensions. The probability of perturbing each coordinate is

controlled by $\varphi(n)$, which is defined as

$$\varphi(n) = \varphi_0\left(1 - \frac{\ln(n - n_0 + 1)}{\ln(N_{\max} - n_0)}\right), \quad n_0 \le n \le N_{\max} - 1, \tag{3}$$

where $\varphi_0 = \min\{20/d, 1\}$. Note that in the definition of φ in the DYCORS algorithm, the probabilities of perturbation on different coordinates are the same. DOSS uses the coordinate knowledge in a local region of x_{best} to determine the perturbation probabilities of different coordinates. This will be discussed in detail in Sect. 4.2.

After perturbation of certain coordinates of the current best point, we generate a pool of candidate points. For each of these candidates, we compute a score. The candidate point with the lowest score is then selected for function evaluation. The score of each candidate point is given by two metrics: a) the RBF metric from the function value of the surrogate at the candidate point and b) the distance metric from the minimum distance between the candidate point and all previously evaluated points. To enhance the current RBF model, we aim to find a candidate point that not only possesses a low RBF surrogate value but also maintains significant distance from points evaluated previously, so a weighted average of the two metrics above is employed as the score that is:

$$\mathcal{W}_n(x) = w_n^R V_n^R(x) + w_n^D V_n^D(x), \tag{4}$$

where $V_n^R(x)$ and $V_n^D(x)$ are the RBF metric value and the distance metric value, respectively. Let Ω_n denote the set of candidate points at iteration n. For each $x \in \Omega_n$, the RBF metric value $V_n^R(x)$ is determined by

$$V_n^R(x) = \begin{cases} \frac{s_n(x) - s_n^{\min}}{s_n^{\max} - s_n^{\min}}, & \text{if } s_n^{\max} \neq s_n^{\min}; \\ 1, & \text{otherwise}, \end{cases} \tag{5}$$

where $s_n^{\max} = \max\{s_n(x) : x \in \Omega_n\}$ and $s_n^{\min} = \min\{s_n(x) : x \in \Omega_n\}$. The distance matrix value $V_n^D(x)$ is given by

$$V_n^D(x) = \begin{cases} \frac{\Delta_n^{\max} - \Delta_n(x)}{\Delta_n^{\max} - \Delta_n^{\min}}, & \text{if } \Delta_n^{\max} \neq \Delta_n^{\min}; \\ 1, & \text{otherwise}, \end{cases} \tag{6}$$

where $\Delta_n(x) = \min_{1 \le i \le n} \|x - x_i\|$, $\Delta_n^{\max} = \max\{\Delta_n(x) : x \in \Omega_n\}$, and $\Delta_n^{\min} = \min\{\Delta_n(x) : x \in \Omega_n\}$. The values of w_n^R and w_n^D are determined through a cyclic weighting strategy. In each iteration, w_n^R is assigned the next value from a predefined weight set Υ in a cyclical manner. Concurrently, w_n^D is defined as the complement to w_n^R, such that $w_n^D = 1 - w_n^R$. In iterations where w_n^R takes a higher value than w_n^D, our algorithm emphasizes exploitation. In iterations where w_n^D takes a higher value than w_n^R, our algorithm emphasizes exploration. Therefore, the cyclic weighting strategy facilitates a dynamic balance between exploration and exploitation across different iterations. In DOSS, the exploration and exploitation are not only controlled by the weight parameters but also affected by the coordinate knowledge.

4 Key features of DOSS algorithm

With a solid understanding of the underlying architecture of general RBF-based optimization algorithms, we now transition to our proposed DOSS algorithm, which introduces three innovative features that enhance performance and solve problems more efficiently.

4.1 New initialization strategy

When dealing with higher-dimensional problems with a limited evaluation budget, the cost of the initialization is considerable. Hence we use the idea in [18] to do the initialization with fewer initial points. The number of initial points in DOSS is set as

$$n_0 = \begin{cases} \lfloor 0.5(d+1) \rceil, & \text{if } d \leq 20, \\ \lfloor 0.4(d+1) \rceil, & \text{otherwise.} \end{cases} \tag{7}$$

where $\lfloor \cdot \rceil$ is to round the float number to the nearest integer value. If $n \leq d+1$, we use the least square solution of linear system (2) to obtain the coefficients of the surrogate model s_n, namely, solve the following system to get λ and c.

$$\begin{pmatrix} \Phi & P \\ P^\top & 0 \end{pmatrix}^\top \begin{pmatrix} \Phi & P \\ P^\top & 0 \end{pmatrix} \begin{pmatrix} \lambda \\ c \end{pmatrix} = \begin{pmatrix} \Phi & P \\ P^\top & 0 \end{pmatrix}^\top \begin{pmatrix} F \\ 0 \end{pmatrix}, \tag{8}$$

If $d+1 \leq n \leq 2(d+1)$, we use multi-quadric[1] kernel $\phi(r) = \sqrt{r^2+\gamma^2}, \gamma > 0$, and constant polynomial g in the RBF surrogate model. In the DYCORS algorithm, a symmetric Latin Hypercube Design is used to generate $2(d+1)$ initial points to initialize the RBF surrogate model.

4.2 New strategy: perturbing coordinates by coordinate knowledge (CK)

In RBF-based algorithms, candidate points are generated by perturbing each coordinate of the current best point. However, as the dimensionality of the optimization problem increases, this perturbation process becomes increasingly complicated. This section unfolds a new strategy we propose to decide which coordinates to perturb, where a concept we refer to as "coordinate knowledge" is defined. Depending on their respective levels of coordinate knowledge, coordinates will be perturbed with different frequencies.

This strategy stems from observations made on the behavior of the DYCORS algorithm on high-dimensional problems. In DYCORS, the probability of perturb-

[1] While the multi-quadric kernel has been used due to its favorable performance in our experiments, we acknowledge that there is no universally optimal RBF surrogate model that applies to all test problems. The effectiveness of a kernel can vary based on the specific characteristics and requirements of the problem at hand. Model selection should be tailored to the unique landscape of each optimization challenge. In line with best practices, we recommend conducting a model selection process to determine the most suitable kernel for your particular settings.

ing coordinates decreases as the number of iterations increases. Consequently, during the iterative implementation of the algorithm, the number of perturbed coordinates (or dimensions) decreases as the number of evaluated points (and iterations) grows. However, within each DYCORS iteration, the perturbation probabilities are identical for all coordinates, regardless of their distinct characteristics. When handling high-dimensional problems, the algorithm might repeatedly perturb only a handful of coordinates, even when the values of the unperturbed coordinates are distant from the global optimal point. As we have limited knowledge of the coordinates that have been less frequently perturbed, the surrogate could guide the search toward evaluating more values of coordinates we already know more about.

To address this, the DOSS algorithm introduces an alternative approach to DYCORS, inspired by the possibility that exploring less frequently perturbed coordinates might further decrease the function value. With this perspective, we adaptively use the information in the local area of the current best point x_{best} to select the coordinate to be perturbed. The procedure for determining the probability of perturbing each coordinate is presented in Algorithm 1.

Algorithm 1 Determine Probability of Perturbing Each Coordinate

Input: weight set $\overline{\Upsilon} \subseteq \Upsilon$, parameter $m \leq n$
Output: probability vector of perturbing each coordinate p_{select}

1: Sort all the evaluated points $x_1, x_2, \ldots, x_n$ and obtain the order $\beta(i)$ such that

$$\text{dist}(x_{\text{best}}, x_{\beta(i)}) \leq \text{dist}(x_{\text{best}}, x_{\beta(j)}) \quad \text{for all } 1 \leq i < j \leq n.$$

2: Calculate the set $S_i = \{x_{\beta(j)}^{\top} e_i : j = 1, \ldots, n\}$, $\mathbf{e}_i = (0, \ldots, 0, 1, 0, \ldots, 0)$ is the unit vector with a 1 at the i-th position. Define $q_i = |S_i|$, $i = 1, \ldots, d$. Hence, q_i is the number of unique values for the i-th component of all the evaluated points. Let $\bar{q}$ be the median value of $\{q_i : i = 1, \ldots, d\}$, $I_p = \{i \in I : q_i < \bar{q}\}$, and $I_r = \{i \in I : q_i \geq \bar{q}\}$, where I is the set of indices of all coordinates. Let $p_{\text{select}} \in [0, 1]^d$ be the probability vector of perturbing each coordinate. Then,

- if $w_n^R \in \overline{\Upsilon}$, then for $i = 1, \ldots, d$:

$$p_{\text{select}}^{(i)} = \begin{cases} \varphi(n), & \text{if } i \in I_p, \\ 0, & \text{otherwise}, \end{cases} \tag{9}$$

- if $w_n^R \notin \overline{\Upsilon}$, then for $i = 1, \ldots, d$:

$$p_{\text{select}}^{(i)} = \begin{cases} \varphi(n), & \text{if } i \in I_r, \\ 0, & \text{otherwise}, \end{cases} \tag{10}$$

3: **return** p_{select}

In Algorithm 1, we use the Euclidean distance in the domain space, denoted as dist, to measure the distance between evaluated points and the current best point. Let denote a reordered sequence $\beta(i), i = 1, ..., n$, of $1, ..., n$ based on the distance to the current best point. We then select the first m elements $x_{\beta(j)}^{\top} e_i$, $j = 1, \ldots, m$, from this reordered sequence. Owing to the stochastic candidate points generation

method, different coordinates are perturbed at different times. Among all evaluated points, we consider the number of unique values evaluated for each dimension i. We define $S_i = \{x_{\beta(j)}^{\top} e_i | j = 1, \ldots, m\}$, where $\mathbf{e}_i = (0, \ldots, 0, 1, 0, \ldots, 0)$ is the unit vector with a 1 at the i-th position. The set S_i consists of all unique values in the i-th coordinate of the evaluated points. If coordinate S_i is perturbed more frequently, the set S_i will contain more elements. Let q_i be the number of elements in S_i, i.e., $q_i = |S_i|, i = 1, \ldots, d$. The term q_i represents the number of unique values in the i-th coordinate of the m closest evaluated points in the region. Consequently, q_i can be seen as the knowledge level for coordinate i.

A viable alternative method for calculating the coordinate knowledge level is using the standard deviation. Given the domain range of various coordinates could differ significantly, an initial step would be to normalize all elements to a range of $(0, 1)$. Let us denote $S_i = |s_1, s_2, ..., s_c|$ as the set of all distinct values in the ith coordinate of evaluated points, denote lb_i and ub_i as the lower and upper bound for the ith coordinate. Subsequently, all elements are normalized according to the function $x \rightarrow \frac{x - \min lb_i}{\max ub_i - \min lb_i}$, for all $x \in S_i$, thereby producing a normalized set. The final step involves calculating the standard deviation of this normalized set, which then serves as the knowledge level.

We can partition all the coordinates into two groups - "rich" and "poor" knowledge groups - based on the above-defined knowledge level. A coordinate is assigned to the "rich knowledge" group if the number of distinct values in the coordinate is greater than the median number of distinct values across all coordinates. Conversely, a coordinate is assigned to the "poor knowledge" group if the number of distinct values within it is less than the median. We denote the index set of coordinates with poor and rich knowledge as I_p and I_r, respectively.

The surrogate model, restricted to the subspace constituted by coordinates with rich knowledge, typically exhibits greater accuracy than one restricted to the I_p subspace. Therefore, the intention of Algorithm 1 is to adjust the perturbation of coordinates to strike a balance between the knowledge levels of different coordinates and the accuracy of the surrogate models within different subspaces in the local region.

As discussed in Sect. 3.3, RBF-based algorithms utilize weights w_n^R and w_n^D to balance exploration and exploitation. In each iteration, w_n^R is assigned the next value from Υ in a cyclical manner. We set Υ as $\{0.3, 0.5, 0.8, 0.95\}$ following the approach used in DYCORS. The selection of this discrete weight set helps maintain a controlled balance between global and local search behaviors, which is crucial for achieving robust optimization results. To further enhance the processes of exploration and exploitation within each iteration, we propose the CK strategy where a subset of Υ: $\overline{\Upsilon} = \{0.3, 0.5, 0.8\}$ is defined. $\overline{\Upsilon}$ partitions the iterations into two scenarios: when $w_n^R \in \overline{\Upsilon}$ and when $w_n^R \notin \overline{\Upsilon}$. When $w_n^R \in \overline{\Upsilon}$, our algorithm emphasizes more on exploration. This corresponds to the iterations where candidate points with larger distances to the current best points are more likely to be selected for evaluation due to the lower weight on the expected surrogate and a reduced emphasis on keeping a large distance from previously evaluated points. Additionally, the CK strategy perturbs coordinates with poor knowledge, thus more candidate points in unexplored regions are generated, which corresponds to Eq. (9). Conversely, when $w_n^R \notin \overline{\Upsilon}$, the algorithm tends to favor exploitation, leading to the selection of candidate points with lower surrogate values.

In these iterations, the CK strategy aims to perturb coordinates with rich knowledge, adding more exploitation to candidate points, which aligns with Eq. (10).

The CK strategy adjusts the perturbation probability according to the knowledge levels across all coordinates. In iterations focusing on exploration, coordinates exhibiting poor knowledge levels are targeted for perturbation, which will generate more candidate points within unexplored regions. Conversely, in iterations with an emphasis on exploitation, coordinates possessing rich knowledge levels are perturbed, thereby generating more candidate points in already explored regions. This strategy, particularly within the context of high-dimensional problems, has the potential to probe a broader range of coordinates and strike a more effective balance between exploration and exploitation in candidate point generation methods. To gain a deeper insight into this strategy, please refer to an illustrative example in Appendix B.

4.3 Generating candidate points using sparse direction search based on the gradient of Surrogate (SG)

This section introduces a new strategy named SG to generate candidate points. Different from most of the existing surrogate-based optimization algorithms (e.g., [21], [20], [22]), where only the surrogate values are utilized, the SG strategy makes use of the gradient information of the surrogate models since it is one of the important pieces of information that can be used in optimization. The search direction with a negative gradient can achieve function value improvement on the current best point. So it would be better if the gradient information from the surrogate model could be utilized in the procedure of candidate generation.

Considering that the strategy is designed to solve higher-dimensional problems, the idea of sparse direction search, which is proven to be efficient (see [22]), is incorporated into the SG strategy. The sparse searching direction set $S(I_{\text{perturb}})$ is defined as

$$S(I_{\text{perturb}}) = \left\{ p \in \mathbb{R}^d : p^\top e_i = 0, \text{ for all } i \in I \setminus I_{\text{perturb}} \right\}. \tag{11}$$

Where $e_i = (0, \cdots, 0, 1, 0, \cdots, 0)$ is the vector with all components equal to 0, except the ith, which is 1. p is the searching direction vector, where all values in the unperturbed coordinates are zero. $I_{\text{perturb}} := \{i \in I : \omega^{(i)} < p^{(i)}_{\text{select}}\}$, where $\omega \in [0, 1]^d$ is a uniform random vector. The superscript (i) represents the ith element of the vector.

The idea of the Sparse direction search based on the Gradient of the surrogate (SG) strategy in DOSS is to combine 1) the sparsity from the dynamic coordinate search used in DYCORS to increase the chance of getting improvement faster with 2) the steepest descent direction of the surrogate function to decrease the objective value faster. For each randomly generated index set I_{perturb}, the closest direction to the steepest descent direction of the surrogate at the current best point is $-\nabla s_n(x_{\text{best}})$, in $S(I_{\text{perturb}})$, which is selected as the random search direction. The detailed procedure for generating candidate points is shown in Algorithm 2.

Algorithm 2 first determines k_1 searching directions. For each direction, k_2 numbers are generated to serve as the step size. So in each iteration, it generates $k = k_1 k_2$ candidate points. The descriptions from line 2 to line 5 are the main part of the algorithm.

Algorithm 2 Generate Candidate Points

Input: current best point x_{best}, surrogate function s_n, the probability vector of perturbing coordinates $p_{\text{select}} \in [0, 1]^d$, standard deviation of the step sizes σ, number of candidate points k, parameters k_1 and k_2 such that $k = k_1 k_2$.

Output: candidate points $y_{n,1}, y_{n,2}, \ldots, y_{n,k}$.

1: **for** $i = 1, \ldots, k_1$ **do**

2: Generate uniform random vector $\omega \in [0, 1]^d$ and I_{perturb}. If $I_{\text{perturb}} = \emptyset$, then set $I_{\text{perturb}} = \{j\}$, where j is uniformly selected at random from I.

3: Let p_i satisfy

$$p_i^{(j)} = \begin{cases} -\nabla s_n^{(j)}(x_{\text{best}}), & \text{if } i \in I_{\text{perturb}}, \\ 0, & \text{otherwise.} \end{cases} \quad j = 1, \ldots, d, \tag{12}$$

where $s_n(x)$ is the surrogate model with n interpolating points. If $\|p_i\| = 0$, then reselect p_i from $S(I_{\text{perturb}})$ randomly such that $\|p_i\| = 1$.

4: Let $\alpha_{\max} := \max\{\alpha \geq 0 : x_{\text{best}} + \alpha p_i \in \mathcal{D}\}$. We then generate k_2 numbers from a truncated normal distribution with mean 0, standard deviation σ, and bounds $[-\alpha_{\max}, \alpha_{\max}]$. These k_2 numbers are represented by α_j for $j = 1, \ldots, k_2$.

5: Let $y_{n,t} := x_{\text{best}} + \alpha_j p_i$, $j = 1, \ldots, k_2$, where $t = (i - 1)k_2 + j$.

6: **end for**

7: **return** candidate points $y_{n,1}, y_{n,2}, \ldots, y_{n,k}$.

Line 2 determines the coordinates to perturb. Line 3 calculates the search direction based on the perturbation index and the gradient of the surrogate function. In perturbed coordinates, the search direction is the negative of the surrogate gradient. While in the unperturbed coordinates, the search direction is zero. Line 4 generates the step size for the search direction above. Multiple step sizes are generated with truncated normal distribution. Line 5 calculates the candidate points based on the search direction and step sizes. For a better understanding of this strategy, please refer to an example in Appendix C.

Note that although Algorithm 2 uses the search direction like the one used in the classical gradient descent method in nonlinear optimization, it is still a random search method. The decision to use random search rather than classical gradient descent is supported by the following aspects. The aim of the gradient descent method is to find a local optimal solution, which is not the same as the goal for global optimization problems, which usually have many local minima. To deal with multiple local minima, random search gives DOSS more chance to escape from the current local optimal area. Note that the step size α_j in Algorithm 2 is not restricted to be positive. Therefore DOSS could possibly find another local optimal area by searching along the "steepest ascent direction". However, the classical gradient descent method may lead to a poor local minimum due to the significant non-convexity of the maximum distance function [18].

4.4 Pseudocode of DOSS algorithm

Previous sections have delved into the specifics of the new strategies employed within DOSS, this section will specifically concentrate on the interplay and integration of

these strategies to form the DOSS algorithm with a comprehensive pseudocode of DOSS provided in Algorithm 3.

Algorithm 3 Directional Optimization Search with Surrogate (DOSS)

Inputs:
1. Objective function f defined over a closed hyper-rectangle $\mathcal{D} \subseteq \mathbb{R}^d$.
2. Maximum number of function evaluations N_{max}.
3. Maximum number of iterations I_{max}.
4. Weight sets Υ and $\overline{\Upsilon}$. $\overline{\Upsilon} \subset \Upsilon$
5. Number of function evaluations for computing the initial surrogate N_S.
6. Number of candidate points generated per iteration N_c.

Output: Best solution x^* within evaluation budget N_{max}.

Step 1. Initialization. Generate n_0 initial points. Add these to the set of evaluated points $\mathcal{A}_n$. Define x_n^* as the current best point in $\mathcal{A}_n$.

Step 2. Fit/Update Surrogate Model. Fit/update surrogate model $s_n(x)$ using evaluated points and their function values, denoted as $\mathcal{B}_n = \{(x_i, f(x_i)) : x_i \in \mathcal{A}_n\}$. (See Section 3.2.)

Step 3.1 Calculate the probability of perturbing each coordinate. Use CK strategy to determine perturbation probabilities for each coordinate based on current weight w_n^R. (See Algorithm 1 in Section 4.2).

Step 3.2 Determine coordinates for perturbation. Generate a uniform random number from [0, 1] for each coordinate. If this value is less than the perturbation probability, perturb the coordinate.

Step 3.3 Perturb the current best point. Given the determined coordinates for perturbation, use the SG strategy to generate candidate points if $w_n^R \in \overline{\Upsilon}$; else use the DYCORS strategy. (See Algorithm 2 in Section 4.3).

Step 4. Select Candidate Points. Choose x_n^E from the candidate points set Ω_n for function evaluation. The selection is based on the lowest score, computed via a weighted sum of surrogate values and the distance to previously evaluated points, as defined in Eq. (4) (See Section 3.3).

Step 5. Evaluate Function. Compute function values at selected points x_n^E.

Step 6. Update Information. Add newly evaluated points x_n^E to the set of evaluated points. Update $\mathcal{A}_{n+1} = \mathcal{A}_n \cup \{x_n^E\}$, $\mathcal{B}_{n+1} = \mathcal{B}_n \cup \{(x_n^E, f(x_n^E))\}$.

Step 7. Check Evaluation Budget. If N_{max} is exhausted, proceed to the next step. Otherwise, return to **Step 2**.

Step 8. Output Best Solution. Return the best solution $x_{N_{max}}^*$ and the search data $\mathcal{A}_{N_{max}}$.

In Step 1, the initial points are sampled with the method described in Sect. 4.1. The number of these points is determined by Eq. (7). Step 2 is executed during each iteration, serving to fit or update the surrogate model, s_n. For further details regarding the RBF model, readers are encouraged to refer to Sect. 3.2. Step 3 is the main component of our algorithm. The results of Step 3 depend on w_n^R, which takes values from Υ in a cyclical manner. During Step 3.1, the probability of perturbation for each coordinate is calculated using Algorithm 1, detailed in Sect. 4.2. This probability is then used in Step 3.2 to select the subset of coordinates for perturbation. In Step 3.3, two strategies are used to perturb the current best point on the determined subset of coordinates. When $w_n^R \in \overline{\Upsilon}$, the SG strategy is adopted to generate candidate points, as outlined in Sect. 4.3. Otherwise, the DYCORS strategy is instead implemented. Subsequently, for each candidate point, we calculate its score in accordance with Eq.(4) in Sect. 3.3. The candidate point with the lowest score is then selected for function evaluation. The algorithm repeats Steps 2 to 6 until the maximum number of function evaluations, as defined by N_{max}, is reached.

To guarantee that the searching strategy in DOSS can cover every corner of the domain, we combine a) the new strategies SG and CK introduced in Sects. 4.2 and 4.3 with b) the dynamic coordinate search method in DYCORS. These strategies a) and b) are called in DOSS alternatively. In a big loop consisting of multiple iterations, DOSS first uses the SG strategy on the part of the coordinates determined by the CK strategy going through each weight in Υ in the first $\left|\overline{\Upsilon}\right|$ iterations. Then in the later iterations, the dynamic coordinate search in DYCORS is used to go through each weight in $\Upsilon \backslash \overline{\Upsilon}$.

5 Convergence property

In this section, we prove the convergence of the DOSS Algorithm. The DOSS algorithm is a hybrid method with multiple strategies, which benefits from the advantages of the new strategies discussed in the above two subsections and simultaneously preserves the same convergence property in DYCORS. The convergence property of the DYCORS algorithm has not been proved before. So before proving the results for the DOSS algorithm, the almost sure convergence proof for the DYCORS algorithm should be proven first.

The proof of the DYCORS algorithm is based on the main theorem in [21]. For the reader's convenience, the main theorem is described below.

Theorem 1 *([21]) Suppose f defined on $\mathcal{D} \subseteq$ compact set $\mathbb{R}^d$ is the objective function to be optimized. x^* is the unique global minimum of f in the domain $\mathcal{D}$. Z_n is the set of candidate points in the n-th iteration and $Z_{n,i}$ is the i-th point in the n-th iteration. And x_n is the evaluated point in the n-th iteration, which is selected deterministically from the values of $\{Z_{n,1}, ..., Z_{n,k}\}_{n\geq 1}$.*
If the random vectors $\{x_n\}_{n\geq 1}$ and $\{Z_{n,1}, ..., Z_{n,k}\}_{n\geq 1}$ satisfy the following conditions

- ***[C1]*** *$Z_{n,1}, Z_{n,2}, ..., Z_{n,k}$ are conditionally independent given the random vectors*

$$\{x_1, ..., x_{n-1}\}, \text{ for } n \geq 2.$$

- ***[C2]*** *For any $j = 1, ..., k$, $x \in \mathcal{D}$ and $\delta > 0$, there exists $a_j(x, \delta) > 0$ such that*

$$P[Z_{n,j} \in (B(x, \delta) \cap \mathcal{D}) | \sigma(\{x_1, ..., x_{n-1}\})] \geq a_j(x, \delta)$$

where, $B(x, \delta)$ is the open ball of radius δ centered as x and $\sigma(\{x_1, ..., x_{n-1}\})$ is the $\Sigma-$field generated by the random vector $\{x_1, ..., x_{n-1}\}$. The x_n^* series below is the best value of the decision vector found over all iterations equal to or less than n. Define a sequence of current best points x_n^*: $x_1^* = x_1$ and $x_n^* = x_n$ if $f(x_n) < f(x_{n-1}^*)$, while $x_n^* = x_{n-1}^*$ otherwise. Then $x_n^* \xrightarrow{a.s.} x^*$.

The following theorem is the main result of this section, which proves the almost sure convergence of the DOSS algorithm.

Theorem 2 *Suppose x^* is the unique global optimizer of f on $\mathcal{D}$ such that $f(x^*) > -\infty$ and $\min_{x\in\mathcal{D}, \|x-x^*\|>\eta} f(x) > f(x^*)$ for all $\eta > 0$. Define the incumbent vectors x_n^* as: $x_1^* = x_1$, $x_n^* = x_n$ if $f(x_n) < f(x_{n-1}^*)$ while $x_n^* = x_{n-1}^*$ otherwise. If*

$\inf_{n\to\infty} \varphi(n) > 0$, *then the incumbent vectors* $x_n^*, n \geq 1$ *generated by DOSS converges to* x^* *almost surely.*

Proof Based on the description in Sect. 5, two new strategies (SG and CK) and the dynamic coordinate search method in DYCORS are called in alternating iterations.

Consider the iteration of the DOSS algorithm, which uses the dynamic coordinate search method in DYCORS. Denote the candidate points of these iterations as $Z_{n,1}^D, Z_{n,2}^D, ..., Z_{n,k}^D$ that are generated independently around current best point. So condition C1 in Theorem 1 is naturally satisfied. Denote the evaluated points of these iterations as $x_1^D, x_2^D, ..., x_{n-1}^D$. Define the current best vectors x_n^{D*} as: $x_1^{D*} = x_1^D$, $x_n^{D*} = x_d^D$ if $f(x_n^D) < f(x_{n-1}^{D*})$ while $x_n^{D*} = x_{n-1}^{D*}$ otherwise.

Based on the dynamic coordinate search, when the variables are perturbed in all coordinates, $Z_{n,j}^D$ has a conditional density (denoted as $h_{n,j}^D$) that follows the truncated normal distribution, which is given by

$$h_{n,j}^D(y|\sigma(\{x_1^D, ..., x_{n-1}^D\})) \tag{13}$$

$$= \frac{(2\pi\sigma_n^2)^{-d/2} \exp\left\{\frac{-\|y - x_{n-1}^{D*}\|^2}{2\sigma_n^2}\right\}}{\int_{\mathcal{D}} (2\pi\sigma_n^2)^{-d/2} \exp\left\{\frac{-\|y - x_{n-1}^{D*}\|^2}{2\sigma_n^2}\right\} dy} \geq C_n > 0 \tag{14}$$

σ_n is the standard deviation of candidate points, C_n is a positive number. Denote the uniform random variable generated for ith coordinate as $r_i, i = 1, ..., d$, p_{select} is the given probability of perturbation of current best point x_n^* in each coordinate. Since the choice of perturbation and normal distribution variable generation are independent, we have the probability that a candidate point is in the ball of radius δ around any point $x \in \mathcal{D}$ is

$$P[Z_{n,j}^D \in (B(x,\delta) \cap \mathcal{D})|\sigma(\{x_1^D, ..., x_{n-1}^D\})] \tag{15}$$

$$\geq \prod_{i=1}^{d} P[r_i < p_{select}] \int_{B(x,\delta)\cap\mathcal{D}} h_{n,j}^D(y|\sigma(\{x_1^D, ..., x_{n-1}^D\}))dy \tag{16}$$

$$\geq \prod_{i=1}^{d} P[r_i < p_{select}] C_n \cdot \mu(B(x,\delta) \cap \mathcal{D}) \tag{17}$$

$$= (p_{select})^d \cdot C_n \cdot \mu(B(x,\delta) \cap \mathcal{D}) > 0 \tag{18}$$

Inequality (16) is because of the DYCORS search strategy. Inequality (17) is because of equality (14) by replacing $h_{n,j}^D(y|\sigma(\{x_1^D, ..., x_{n-1}^D\}))$ with C_n. μ is Lebesgue measure so that $\mu(B(x,\delta) \cap \mathcal{D}) = \int_{B(x,\delta)\cap\mathcal{D}} dy$. Equality (18) holds for $r_i, i = 1, ..., d$. r_i is uniform distributed random variable in [0, 1]. So Condition C2 in Theorem 1 statement is also verified. Then based on Theorem 1, the defined sequence $x_n^{D*} \xrightarrow{a.s.} x^*$.

According to the definition of x_n^*, for any sample point ω, we have $f(x^*) \leq f(x_n^*(\omega)) \leq f(x_n^{D*}(\omega))$. It implies that $\{\omega : f(x_n^{D*}(\omega)) - f(x^*) < \epsilon\} \subset \{\omega :$

$f(x_n^*(\omega)) - f(x^*) < \epsilon\}$, for any $n \in \mathbb{N}, \epsilon > 0$. So when we consider the limit of the sequence x_n^*, we have

$$\{\omega : \lim_{n\to\infty} f(x_n^{D*}(\omega)) = f(x^*)\} \subset \{\omega : \lim_{n\to\infty} f(x_n^*(\omega)) = f(x^*)\}$$

Then we have the probability relationship

$$1 = P\{\omega : \lim_{n\to\infty} f(x_n^{D*}(\omega)) = f(x^*)\} \tag{19}$$

$$\leq P\{\omega : \lim_{n\to\infty} f(x_n^*(\omega)) = f(x^*)\} \tag{20}$$

x_n^{D*} is generated with DYCORS strategy. The equality (19) is because $x_n^{D*} \xrightarrow{a.s.} x^*$, which is proved in Theorem 1. According to the property of probability measure, we have

$$0 \leq P\{\omega : \lim_{n\to\infty} f(x_n^*(\omega)) = f(x^*)\} \leq 1 \tag{21}$$

So based on (19)-(21), we have

$$P\{\omega : \lim_{n\to\infty} f(x_n^*(\omega)) = f(x^*)\} = 1$$

So $f(x_n^*) \xrightarrow{a.s.} f(x^*)$. By the uniqueness of global optimum, $x_n^* \xrightarrow{a.s.} x^*$. □

6 Computational experiments

In this section, we evaluate the computational performance of DOSS on some higher dimensional box-constrained black-box optimization problems. DOSS is implemented in Python and is based on the pySOT package [7]. All experiments in this section are run on the HPC named ASPIRE 1 from NSCC. Each node of the HPC has an Intel E5-2690v3 2.60GHz CPU and 128GB memory installed.

6.1 Alternative Optimization Methods

We compare our DOSS algorithm with the original DYCORS algorithm, the serial RBF-based algorithm in RBFOpt package [18], the Mesh Adaptive Direct Search (MADS) with the metaheuristic Variable Neighborhood Search (VNS) algorithm (MADS-VNS) [1], Genetic Algorithm (GA), Evolution Strategy with Covariance Matrix Adaptation (CMA-ES) [12] [13], and TuRBO [8].

DYCORS algorithm is an RBF-based optimization algorithm that incorporates an idea from the Dynamically Dimensioned Search (DDS) algorithm. We use the implementation in pySOT package [7] for experiments.

The RBFOpt software toolbox implements two algorithms: the Metric Stochastic Response Surface Method (MSRSM, the precursor of DYCORS) [21] and Gutmann's

RBF algorithm [11]. It has been shown that the MSRSM performs much better than Gutmann's RBF algorithm, especially on higher dimensional problems [21]. Therefore, we only compare with the MSRSM version in RBFOpt software. The MSRSM algorithm in RBFOpt also uses the restart strategy and the distance weighting features in the earlier paper [21]. RBFOpt also has some new strategies to enhance the MSRSM algorithm, like the refinement step, restoration step and automatic model selection, and selecting the center of the domain as an initial evaluation point. These strategies are proposed in [18].

The MADS-VNS [1] algorithm combines MADS with the VNS algorithm. The MADS algorithm is based on the generalized pattern search algorithm, which can converge to the local optimum. VNS algorithm helps escape from the local optimum with a random perturbation method. MADS-VNS is available in the NOMAD software [16].

GA is a population-based heuristic method, which is inspired by the process of natural selection.

CMA-ES [12] [13] adapts arbitrary normal mutation distributions in evolution strategies (ES), which has proved to be more efficient than other ES-based algorithms.

TuRBO [8] uses the trust region strategy for Bayesian optimization and allows multiple local optimization runs using independent probabilistic models. Global optimization is achieved in TuRBO's framework by maintaining several local Gaussian process surrogate models simultaneously. Samples in different trust regions are allocated by an implicit multi-armed bandit approach.

6.2 Test instances and experimental setup

We evaluated the performance of DOSS using a test set of 14 multi-modal functions, including Ackley, Branin, Eggholder, Hartmann3, Hartmann6, Keane, Levy, Michalewicz, Rana, Rastrigin, Schubert, Schwefel, Styblinski-Tang, and Weierstrass. These functions were expanded to 36, 48, and 60 dimensions, culminating in 42 higher-dimensional test problems. These test functions are categorized based on their dimensional flexibility: the first category consists of functions adaptable to any dimension due to well-defined mathematical expressions, facilitating straightforward extensions to higher dimensions. Conversely, the second category, includes functions–specifically Branin, Hartmann3, and Hartmann6–whose mathematical formulations are confined to certain dimensions.

To accommodate the latter's dimensional restrictions, we employ a wrapper function approach. This method allows for dimension-agnostic input handling. Inputs of higher dimensions are segmented into 'buckets', each corresponding in size to the original function's dimensionality. The wrapper then calculates the average value across these buckets, effectively extending the function to the desired higher dimensions while preserving the integrity of the base function's properties. For a detailed explanation and a practical example of extending the Branin function to higher dimensions using this wrapper function method, please see Appendix D.

The wrapper function method, while effective, simplifies interactions by replicating base instances. We recognize that this could influence algorithm performance by

favoring strategies that exploit the resulting sparsity. The method was applied to only three of the fourteen test functions. The majority of the test functions used in our experiments inherently support higher dimensions without the need for a wrapper function, thus providing a fair and balanced testbed for assessing the performance of the algorithms under consideration. Furthermore, we have meticulously ensured that problems are extended only to dimensional multiples, effectively preventing the exclusion of any decision variables.

For each test problem, we set an optimization budget proportional to the dimensionality of the problem, specifically $50(d+1)$. All algorithms were then applied to each test problem across 30 trials, each initialized with a unique random seed to ensure the robustness of our results. It is important to note that each algorithm was initialized with its own set of randomly generated initial points, which could influence the comparability and reproducibility of the results. This approach allows each algorithm to demonstrate its inherent strengths and adaptability under varied initial conditions.

To address potential concerns regarding variance and to enhance the reproducibility of our findings, we have included an analysis of the variance in the objective function values of the optimal solutions over these 30 trials. This analysis, presented in Appendix 1, offers a detailed comparison of the average variance over three dimensional scenarios across different algorithms and problems, thereby providing a clearer view of the stability and reliability of each algorithm under test conditions.

All the algorithms are also tested on two real-world optimization problems, robot pushing and rover trajectory planning problems. The robot pushing problem is a 14-dimensional control parameter tuning problem for robot pushing ([24]). The rover trajectory planning problem ([8]) is a 60-dimensional problem to optimize a reward function for the trajectory of a 2D rover. The optimization budget on each instance is set to 3000.

The values of l_i and r_i for each test problem are given below. Table 1 lists the lower and upper bounds of the domains (hypercubes) of the 13 multi-modal test functions. The domain of the test function Branin is $[-5, 10] \times [0, 15]$. For the robot pushing problem, the value of l_i and r_i for each dimension is as shown in Table 2. The domain of the Rover trajectory planning problem is a hypercube with $l_i = 0$ and $r_i = 1$.

In our test, DOSS implements our new initialization strategy described in Sect. 4.1 and the hybrid candidate points generation method introduced in Sect. 5. The initial points of DOSS are generated by a Latin hypercube design maximizing the minimum distance between points [18]. Weight set $\overline{\Upsilon}$ and parameter m in Algorithm 1 are set to $\{0.3, 0.5, 0.8\}$ and $2(d+1)$, respectively. We have implemented two versions of the DOSS algorithms, namely DOSS(cnt) and DOSS(std). The difference between these two versions lies in their respective definitions of knowledge levels as outlined in Sect. 4.2. DOSS(cnt) defines the knowledge level as the count of unique values for each coordinate. On the other hand, DOSS(std) calculates the knowledge level by determining the standard deviation after the values of the evaluated points in each coordinate have been suitably scaled. The number of candidate points k in each iteration is set to $100d$ for any dimension d. Parameters k_1 and k_2 in Algorithm 2 are set to d and 100, respectively. All the other parameter settings are the same as the default ones in DYCORS. Settings of DYCORS, RBFOpt, and TuRBO are all set to default or recommended values in their papers or codes.

Table 1 The lower and upper bounds of domains of the 13 multi-modal test functions

Function	l_i	r_i
Ackley	-10	20
Eggholder	-512	512
Hartman3	0	1
Hartman6	0	1
Keane	1	10
Levy	-5	15
Michalewicz	0	π
Rana	-500	500
Rastrigin	-3.12	8.12
Schubert	-10	10
Schwefel	-512	512
Styblinski-Tang	-5	5
Weierstrass	-3	8

Table 2 The value of l_i and r_i of each dimension for the robot pushing problem

l_i	-5	-5	-10	-10	2	0	-5	-5	-10	-10	2	0	-5	-5
r_i	5	5	10	10	30	2π	5	5	10	10	30	2π	5	5

6.3 Tools for result analysis

In the numerical experiments, we test our algorithm by comparing it to multiple benchmark algorithms across a wide range of problems. To present a vast array of results within a limited space, we use the performance profile and data profile [6, 17] to compare algorithm performances. These profile plots assess the overall performance of algorithms on a given test set, where a higher curve signifies that an algorithm is capable of solving a larger proportion of test problems to a given accuracy level.

The accuracy parameter, denoted as τ, is defined within the context of the following convergence test:

$$f(x_0) - f(x) \geq (1 - \tau)(f(x_0) - f_L), \tag{22}$$

where $\tau > 0$ serves as a tolerance threshold, while x_0 represents the initial starting point for the problem. f_L is computed for each problem $p \in P$ as the lowest value of f achieved by any algorithm within a specified number, μ_f, of function evaluations.

Performance profiles primarily highlight relative performance in comparison to the best-performing algorithm. The X-axis in the performance profile represents the performance ratio, defined as the ratio of the computing time (or number of function evaluations) of the current algorithm to that of the best-performing algorithm, provided that they both meet the convergence criteria. It is worth noting that while performance profiles offer clear comparisons between two solvers on a given set (higher curves indicate better results) when comparing more than two solvers, they can determine the

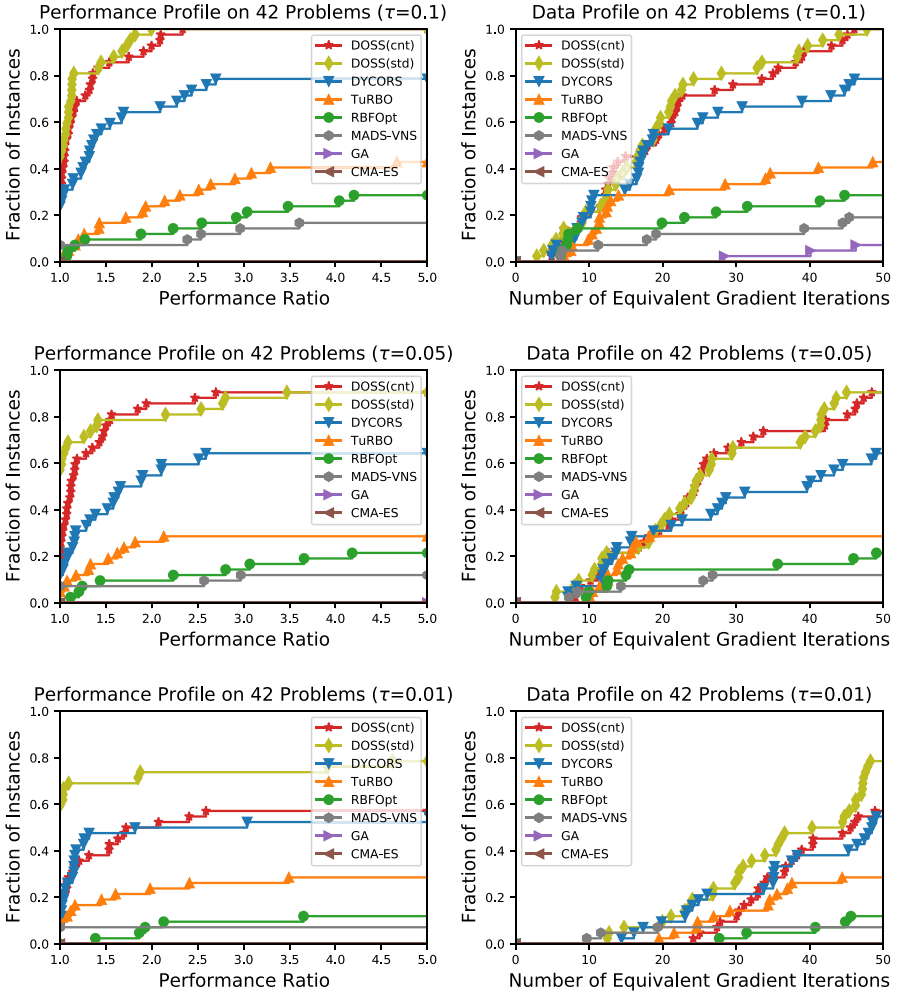

Fig. 1 Comparison of DOSS with DYCORS, TuRBO, MADS-VNS, CMA-ES, GA, and RBFOpt using performance profiles (left column) and data profiles (right column) for three different accuracy levels: $\tau = 0.1$ (top row), 0.05 (middle row), and 0.01 (bottom row). In the data profiles, higher curves indicate better performance, while in the performance profiles, the highest curve represents the best result. Detailed explanations of these profile plots are provided in Sect. 6.3. Note that in some plots, certain curves (e.g., GA, CMA-ES) may be missing, appearing at the bottom line

best solver but may not adequately assess the relative performance of the remaining solvers [10].

Data profiles, on the other hand, place a greater emphasis on changes in performance with respect to the number of evaluations. The X-axis in the data profile denotes the number of equivalent gradient iterations, which can be perceived as the number of evaluations adjusted for problem dimensions. Higher curves in the data profiles indicate better results when comparing solvers.

6.4 Comparison of different algorithms

Figure 1 shows the performance and data profiles of the eight algorithms. On all three accuracy levels, DOSS outperforms the other three methods on the test set. Both DOSS and DYCORS perform better than the other two methods. For the 5% accuracy level, DOSS reaches the best function value on over 90% instances, while less than 20% cases are achieved by MADS-VNS. Although DYCORS converges faster than DOSS in the early stage (the middle and right subfigures in the lower pane in Fig. 1), the difference is only about 5% (the left subfigure in the lower pane in Fig. 1). DOSS converges much faster than the other three methods on the equivalent gradient iterations 45 to 50.

In Fig. 2, the test problems are grouped based on the dimensionality (36, 48, 60). The performance and data profiles are plotted for each group of test problems. In data profiles, higher curves indicate better results, while in performance profiles, the highest curve indicates the best results. The accuracy level is set as 5%, meaning the problem is considered solved if the solution is within 5% of the true answer. As dimensionality increases, our proposed DOSS algorithm can solve a greater number of problems to the given accuracy level. Furthermore, the superiority of DOSS over other algorithms becomes increasingly significant.

Figure 3 presents the comparison of the DOSS algorithm with or without fewer initial points (randomly selected by Latin Hypercube sampling) strategy. DOSS (SLHD) means the DOSS algorithm uses the Symmetrtic Latin Hypercube method for initial points sampling. In most cases, DOSS with fewer initial points strategy is slightly better than the DOSS without this strategy.

Figure 4 compares the DOSS algorithm with or without Perturbing coordinate by Coordinate Knowledge Strategy (CK) and Generating Candidates Using Sparse Direction Search Based on The Gradient of Surrogate Strategy (SG) strategies (described in Sect. 4.3). The experiment is designed to assess the relative importance of CK and SG strategy tested on 43 problems (Sect. 6.2). The curves "CK" means the algorithm only has CK strategy. "SG" means the algotithm only has a SG strategy. The profile plots show that The DOSS algorithm works better when using both of the strategies. Results imply DOSS is the best of the three algorithms when many evaluations are done, and CK alone is more helpful than SG alone, especially when fewer evaluations are done.

Figure 5 compares the DOSS algorithm with other algorithms on real-world problems. In the 14D Robot pushing problem, the proposed DOSS algorithm converges quickly and successfully to find the best solutions among all the algorithms. In the 60D Rover trajectory planning problem, most of the algorithms can find a satisfactory solution after 3000 evaluations. However, the DOSS algorithm can find this solution with less than 1000 function evaluations.

In most test problems, the proposed DOSS algorithm has the best performance under different accuracy levels. Its advantage becomes more significant as the problem dimension increases. The comparisons of different algorithm ingredients (CK, SG) justify the design of the proposed algorithm. The applications in machine learning problems prove the effectiveness of DOSS in real problems.

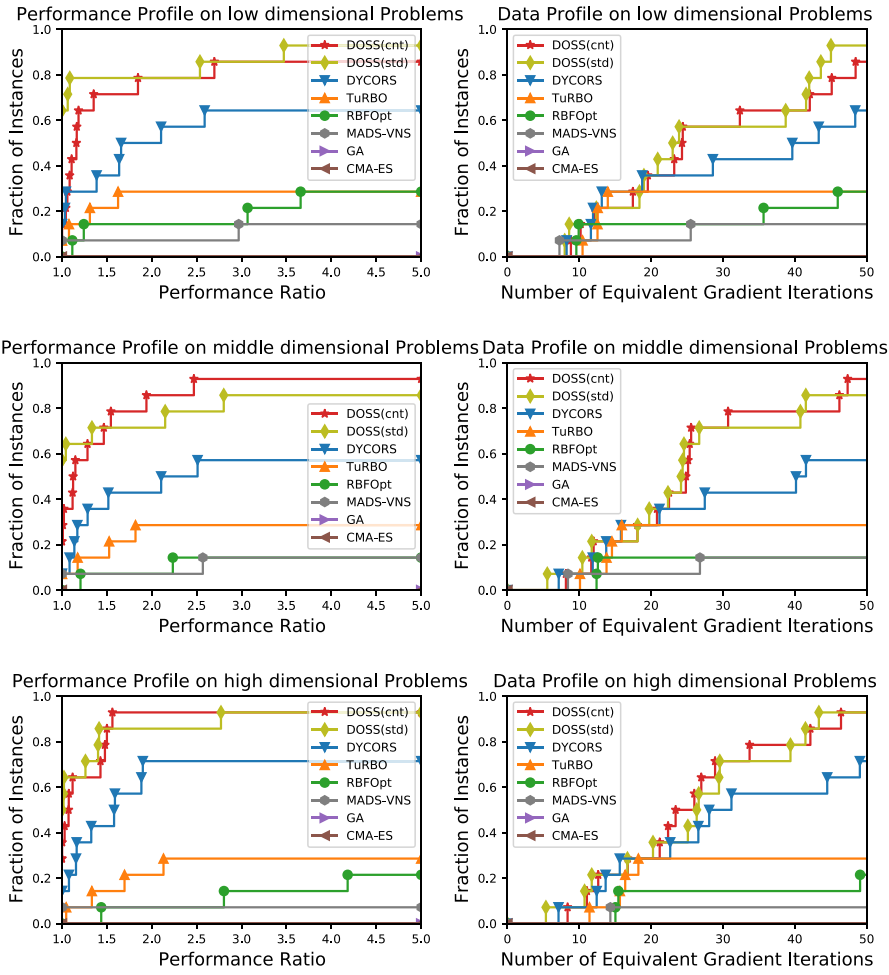

Fig. 2 Comparison of DOSS with DYCORS, TuRBO, MADS-VNS, CMA-ES, GA, and RBFOpt using performance profiles (left column) and data profiles (right column) for accuracy level $\tau = 0.05$. The figures represent results for problems of different dimensions: 36 (top row), 48 (middle row), and 60 (bottom row). In the data profiles, higher curves denote better performance, while in the performance profiles, the highest curve indicates the best result. Detailed explanations of these profile plots are provided in Sect. 6.3. Note that in some plots, certain curves (e.g., GA, CMA-ES, MADS-VNS) may be missing and are located at the bottom line

7 Conclusions

We propose a new algorithm, DOSS, with novel features that has a better performance on higher (60) dimensional problems than other leading serial surrogate global optimization algorithms. Since we are targeting expensive objective functions, surrogate algorithms are expected to be much more efficient than non-surrogate algorithms at achieving good answers with fewer expensive objective function evaluations. DOSS differs from the earlier surrogate algorithms DYCORS in several ways, but the most

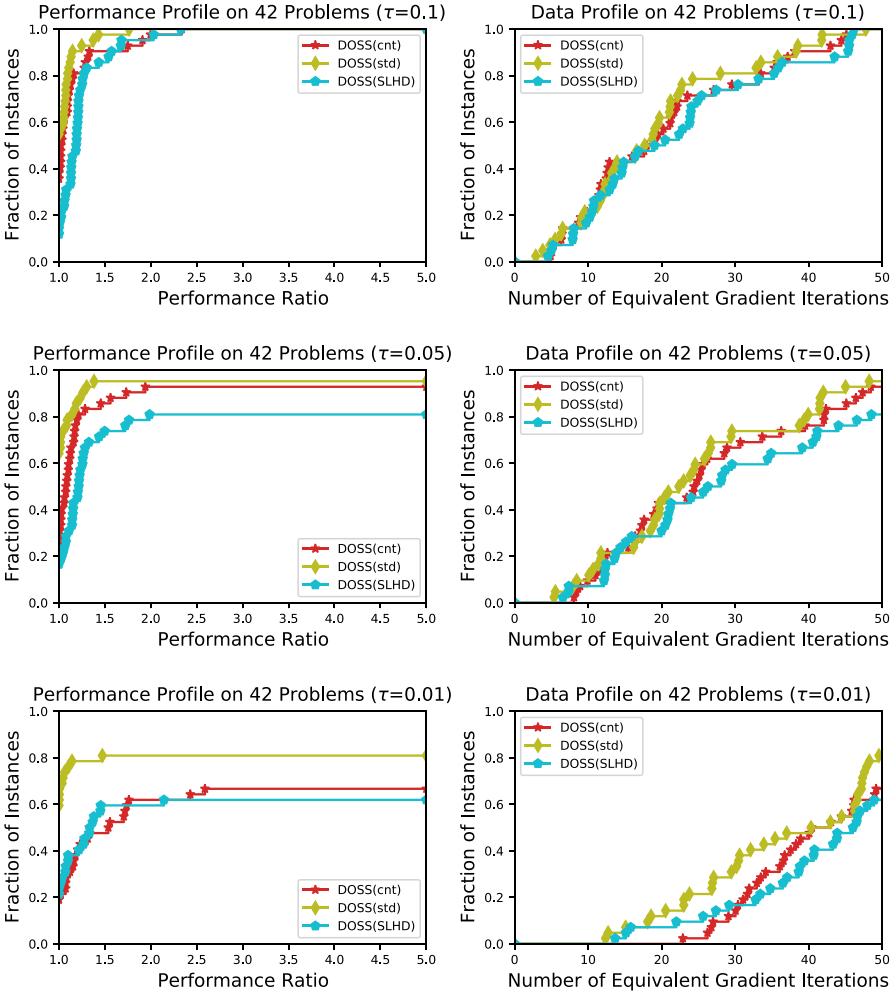

Fig. 3 Comparison of DOSS with and without the fewer initial points strategy using performance profiles (left column) and data profiles (right column) for accuracy levels $\tau = 0.1$ (top row), 0.05 (middle row), and 0.01 (bottom row). DOSS uses the fewer initial points strategy, while DOSS (SLHD) employs the Symmetric Latin Hypercube Design method for initial point sampling. In the data profiles, higher curves denote better performance, whereas in the performance profiles, the highest curve represents the best result. A detailed explanation of these profile plots is provided in Sect. 6.3

important is that a new metric "coordinate knowledge" influences the choice of the next evaluation point. In the DOSS algorithm, three new strategies are developed to help to solve higher dimensional problems quickly. (1) Coordinate Knowledge (CK) is used to encourage perturbations in coordinates (CK) that in earlier evaluations previously were less frequently changed; (2) Fewer initial points are used to reduce the inefficient initial sampling; and (3) Sparse Directional Search Based on the Gradient of Surrogate (SG) is used to achieve faster improvement. Extensive numerical results

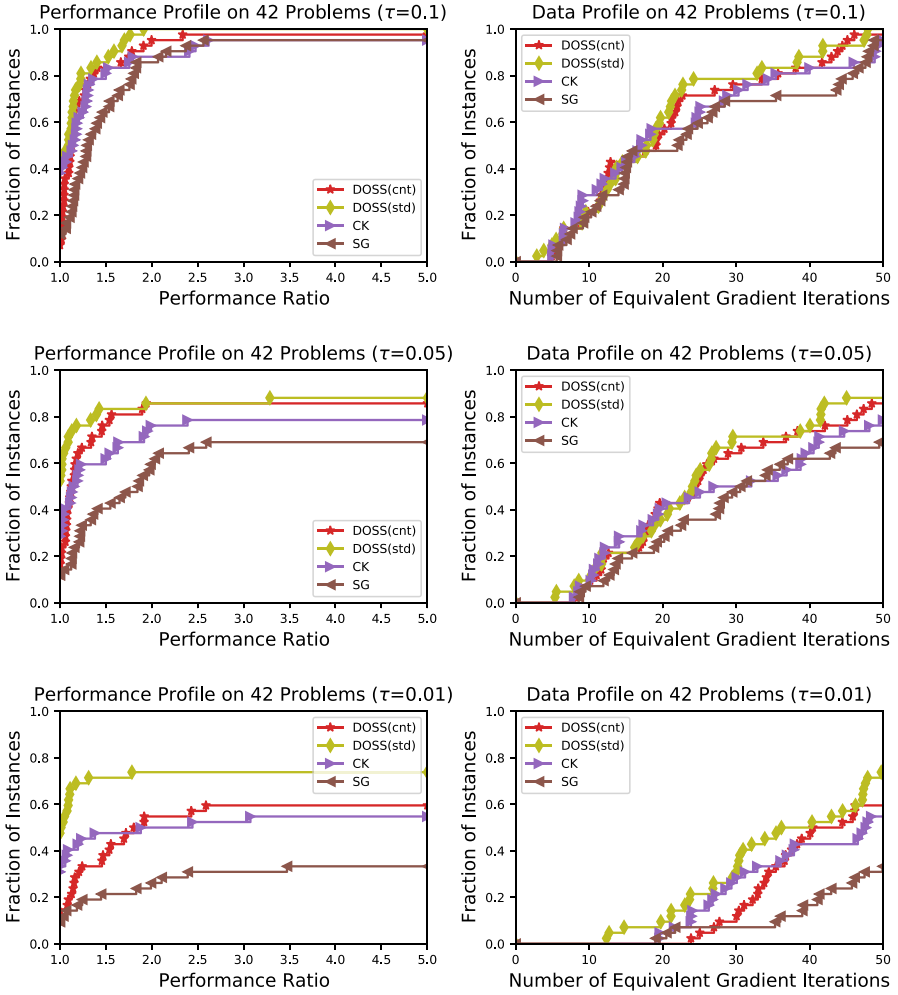

Fig. 4 Comparison of DOSS with and without the CK and SG strategies using performance profiles (left column) and data profiles (right column) for accuracy levels $\tau = 0.1$ (top row), 0.05 (middle row), and 0.01 (bottom row). In the data profiles, higher curves indicate better performance, while in the performance profiles, the highest curve represents the best result. A detailed explanation of these profile plots is provided in Sect. 6.3. This experiment evaluates the relative efficacy of the CK and SG strategies across 42 problems, with details provided in Sect. 6.2. Here, 'DOSS' refers to the implementation that incorporates both CK and SG strategies. The label 'CK' denotes the DOSS algorithm using only the CK strategy, while 'SG' represents the DOSS algorithm using only the SG strategy. The results indicate that DOSS, incorporating both strategies, outperforms the other configurations when a higher number of evaluations are conducted. Furthermore, the CK strategy alone proves to be more effective than the SG strategy when the number of evaluations is limited

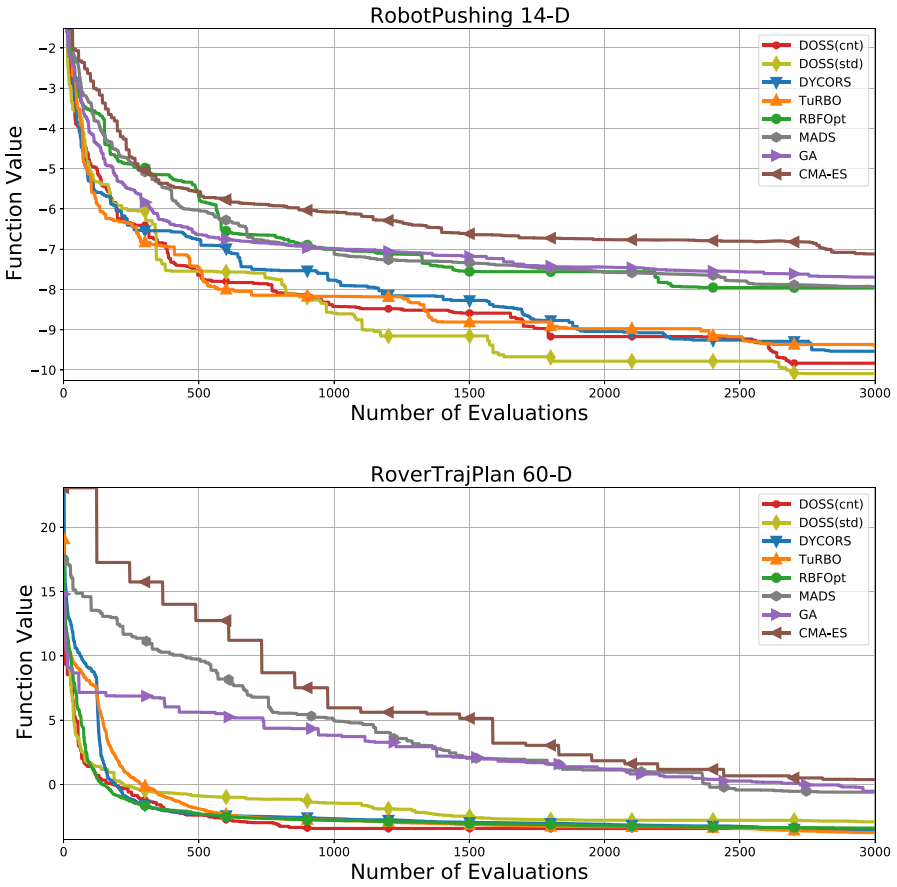

Fig. 5 Progress plots for optimization on two problems: 14-dimensional Robot Pushing (top) and 60-dimensional Rover Trajectory Planning (bottom). The x-axis shows the number of iterations, while the y-axis indicates the objective function value. The plots illustrate a decreasing trend in objective function values, reflecting improvements in solution quality over iterations. Algorithms with lower curves represent better performance"

show that each individual new strategy improves the performance of the algorithm, and the combination of the three strategies gives excellent results.

A Notations and abbreviations

Table 3 Notations

Notations	Descriptions	Section or equation of first appearance
$f(x)$	Continuous objective function	Sect. 2
$\mathcal{D}$	Closed hypercube in $\mathbb{R}^d$	Sect. 2
n_0	Number of initial points	Sect. 3.1
$N_{\max}$	Maximum number of function evaluations	Sect. 3.1
$s_n(x)$	The surrogate model fitted by n points	Sect. 3.2, Equation (1)
λ_i	The coefficients of radial basis function	Sect. 3.2, Equation (2)
c_j	The coefficients of linear polynomial	Sect. 3.2, Equation (2)
$\varphi(n)$	The probability of perturbing each coordinate	Sect. 3.3, Equation (3)
$\mathcal{W}_n(x)$	Weighted average of the RBF metric value and the distance metric value.	Sect. 3.3, Equation (4)
$V_n^R(x)$	The RBF metric value	Sect. 3.3, Equation (4)
$V_n^D(x)$	The distance metric value	Sect. 3.3, Equation (4)
$dist()$	Euclidean distance in the domain space	Sect. 4.2
$\beta(i)$	Reordered index of x_i based on the distance	Sect. 4.2
I_p	The index set of dimension with poor knowledge	Sect. 4.2, Equation (9)
I_r	The index set of dimension with rich knowledge	Sect. 4.2, Equation (10)
Υ	Weight set for candidate points selection	Sect. 4.2
S_i	The perturbation value set of coordinate i for all evaluated points.	Sect. 4.2
q_i	The number of elements in set S_i	Sect. 4.2
p_{select}	Probability vector of perturbing each coordinate	Sect. 4.2, Equation (9)
$S(I_{perturb})$	The sparse searching direction set	Sect. 4.3, Equation (11)
e_i	The vector with all components equal to 0,except the ith, which is 1	Sect. 4.3, Equation (11)
Superscript D ($e.g.$, Z_n^D, x_n^D)	Values related to the DYCORS iteration of DOSS algorithm	Sect. 5, Theorem 2
Abbreviations	Descriptions	
DOSS	Directional Optimization Search with Surrogate	Sect. 1
CK	Coordinate knowledge	Sect. 4.2
SG	Sparse direction search based on the gradient of surrogate.	Sect. 4.3

B Algorithm example 1

To make the explanation of Algorithm 1 more understandable, we provide the following example and calculate the defined variables step by step.

Example 1 Set weight set $\overline{\Upsilon} = \{0.95\} \subseteq \Upsilon = \{0.3, 0.5, 0.8, 0.95\}$, parameters $m = 2, n = 4$, $x_{\text{best}} = (1, 2, 10, 4)$ and evaluated points $x_1 = (1, 2, 3.2, 4)$, $x_2 = (1, 5.3, 7, 8.1)$, $x_3 = (2, 2, 3.2, 4)$, $x_4 = (6, 9, 10, 8.2)$.

In this iteration, algorithm 1 calculates the following steps:

1. Calculate the distance between x_{best} and x_1 to x_4.

$$dist(x_1, x_{\text{best}}) = 6.80, dist(x_2, x_{\text{best}}) = 6.06,$$
$$dist(x_3, x_{\text{best}}) = 6.87, dist(x_4, x_{\text{best}}) = 9.57,$$

   and select 2 points with the lowest distance. $x1, x2$.
2. Calculate the set S_i, $S_1 = \{x_1^T e_1, x_2^T e_1\} = \{1\}$, $S_2 = \{2, 5.3\}$, $S_3 = \{3.2, 7\}$, $S_4 = \{4, 8.1\}$. (Since S is a set, the duplicated elements will be removed). Calculate q, $q_1 = 1, q_2 = 2, q_3 = 2, q_4 = 2$. The median $\bar{q}$ of $q1$ to $q4$ is 2. Since $q_2, q_3, q4 >= \bar{q}$, based on the definition of I_p and I_r, $I_r = 2, 3, 4$, $I_p = 1$.
3. The weight w_n^R cycle in $\Upsilon = \{0.3, 0.5, 0.8, 0.95\}$. When w_n^R is in $\overline{\Upsilon}$, for index set I_r(dimension 1), the algorithm set $p_{\text{select}} = 0$. When w_n^R is not in $\overline{\Upsilon}$, for index set I_p (dimension 2, 3, 4), the algorithm set $p_{\text{select}} = 0$.

Please note that the continuous variables are considered the same as the integer variables.

C Algorithm example 2

For a better explanation of Algorithm 2, we provide the following example and calculate the defined variables step by step.

Example 2 In the nth iteration, given the feasible region $\mathcal{D} = [-10, 10]^8$, the current best point $x_{\text{best}} = (1, 2.2, 3, 4, 4.2, 5.6, 6.4, 7.2)$, the gradient vector of surrogate model at current best point $\nabla s_n = (1.1, -2, -3, -2.1, -4, 5.4, 6.2, 8)$ and the set of perturbed coordinates is $I_{\text{perturb}} = \{1, 3, 5, 7\}$

In the nth iteration, for the kth candidate point, Algorithm 2 calculates the following steps:

1. Calculate the search direction $p_k = (-\nabla s_n^{(1)}, 0, -\nabla s_n^{(3)}, 0, -\nabla s_n^{(5)}, 0, -\nabla s_n^{(7)}, 0)$ $= (-1.1, 0, 3, 0, -4, 0, -6.2, 0)$.
2. Calculate the step size α_{max} such that the new points are in the feasible region. It can be obtained by calculating the maximum step size (to stay within the feasible region) for each coordinate and then calculating the minimum of all coordinates. $\alpha_{max} = \min(8.18, 6.1, 4.3, 6.67, 3.55, 0.81, 0.58, 0.35) = 0.35$.
3. Generate k_2 numbers from a truncated normal distribution with mean 0, standard deviation σ and bounds $[-\alpha_{max}, \alpha_{max}]$

4. Calculate k_2 candidate points, $y_{n,t} = (1 - 1.1\alpha_j, 2.2, 3 + 3\alpha_j, 4, 4.2 - 4\alpha_j, 5.6, 6.4 - 6.2\alpha_j, 7.2)$, $j = 1, \ldots, k_2$ where $t = (k-1)k_2 + j$.

D Higher-dimensional function extension via wrappers

To elucidate the methodology for extending test functions to higher dimensions using a wrapper function, consider the following detailed example:

Example 3 *Extending the Branin Function to 24 Dimensions*
The Branin function, originally defined in 2 dimensions, can be accessed for its mathematical expression at SFU's Optimization Benchmarks (https://www.sfu.ca/%7Essurjano/branin.html). To adapt the Branin function for a 24-dimensional input while preserving its characteristics, we employ a wrapper function that operates as follows:

1. **Input Grouping**: The 24-dimensional input vector is divided into 12 pairs (buckets), grouping every two consecutive dimensions together. This step ensures that we can apply the 2-dimensional Branin function to each pair, treating them as independent instances of the 2-dimensional problem.
2. **Function Evaluation**: For each of the 12 groups, the Branin function is evaluated using the corresponding pair of dimensions as input variables. This yields 12 function values, each reflecting the outcome of the Branin function for a specific subset of the input vector.
3. **Aggregation**: The final output of the wrapper function is calculated as the average of these 12 Branin function values. This averaging process produces a single scalar value that represents the extended function's output for the 24-dimensional input.

By averaging the outcomes across different groups, the wrapper function effectively scales the dimensionality of the base function while retaining its core computational features. This approach allows us to generalize the application of fixed-dimension functions like the Branin function to higher-dimensional optimization problems.

E Comparative analysis of solution stability across algorithms

This section presents a compilation of the average variance in the objective function value of the best solution across 30 trials, as displayed by various algorithms for each test function. Table 4 reflects an average taken across three different dimensional scenarios for each problem-36, 48, and 60-to furnish a holistic view of the variance observed.

The data elucidates that while the average variance of the objective function value for our DOSS algorithm isn't invariably the lowest, it consistently maintains a comparable and competitive performance across all test functions. This indicates a robust general performance, irrespective of the test problem. It's noteworthy that for certain problems (i.e., Keane, Levy, Rastrigin, Schwefel), the DOSS algorithm demonstrates a significantly low variance, underscoring its potential as a stable and reliable optimization strategy in diverse scenarios. Conversely, in instances where the DOSS algorithm

Table 4 Comparative average variance of optimal solutions across algorithms and problems

Algorithm	DOSS(cnt)	DOSS(std)	DYCORS	RBFOpt	TuRBO
Ackley	0.035913165	0.030188345	0.000276936	0.061542777	0.036019874
Branin	0.002294102	0.001172604	0.000242631	0.010415402	0.000682201
Eggholder	1655.800089	1057.342766	1648.148028	5013.241693	1048.854895
Hartman3	0.003253034	0.00389706	0.002738283	0.036637101	0.001545971
Hartman6	0.00054266	0.000517736	0.000354844	0.031546261	0.000489476
Keane	8.69088E-06	1.217E-05	0.000170966	0.000274416	0.000828288
Levy	2.018042393	0.229723007	1.544130124	335.2922978	7.808223596
Michalewicz	3.041278791	2.869501601	1.684024865	6.024584154	3.743543101
Rana	310.1689703	278.6340097	377.5568787	2350.846772	216.6758447
Rastrigin	132.3887153	71.62749015	356.1077981	2722.296652	1223.857615
Schubert	1102.876879	1940.431702	2145.306045	0.443261489	1047.771936
Schwefel	335870.1491	278297.1207	801900.9073	2808207.758	471484.0024
StyblinskiTang	1512.059649	1755.997712	1283.288604	1964.552273	1549.415589
Weierstrass	0.083030024	0.045896489	0.041303207	17.76563085	3.037263192

exhibits a higher variance, it remains within a tight range of the best-performing algorithms, suggesting that DOSS can sustain performance consistency, which is crucial for practical optimization tasks.

Acknowledgements The authors wish to thank the editors and three anonymous reviewers for carefully reading our paper and giving constructive suggestions. Their comments have really helped to improve the paper.

Author contributions All authors contributed to the study conception and design. All authors read and approved the final manuscript.

Funding This project was supported by start-up funds from the National University of Singapore for the Distinguished Professorship of Christine A. Shoemaker.

Data availability statement The datasets generated and/or analyzed during the current study are available in the GitHub repository: DOSS-Algorithm on GitHub (https://github.com/limengliu95/DOSSAlgorithm).

Code Availability The implementation of DOSS algorithm is open source and available for download, https://doi.org/10.5281/zenodo.13731558.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Audet, C., Béchard, V., Le Digabel, S.: Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search. Journal of Global Optimization **41**(2), 299–318 (2008)
2. Binois, M., Ginsbourger, D., Roustant, O.: A warped kernel improving robustness in bayesian optimization via random embeddings. In: International Conference on Learning and Intelligent Optimization, pp. 281–286. Springer (2015)
3. Binois, M., Ginsbourger, D., Roustant, O.: On the choice of the low-dimensional domain for global optimization via random embeddings. Journal of global optimization **76**(1), 69–90 (2020)
4. Björkman, M., Holmström, K.: Global optimization of costly nonconvex functions using radial basis functions. Optimization and Engineering **1**(4), 373–397 (2000)
5. Chen, B., Castro, R., Krause, A.: Joint optimization and variable selection of high-dimensional gaussian processes. arXiv preprint arXiv:1206.6396 (2012)
6. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Mathematical programming **91**(2), 201–213 (2002)
7. Eriksson, D., Bindel, D., Shoemaker, C.A.: pysot and poap: An event-driven asynchronous framework for surrogate optimization. arXiv preprint arXiv:1908.00420 (2019)
8. Eriksson, D., Pearce, M., Gardner, J., Turner, R.D., Poloczek, M.: Scalable global optimization via local Bayesian optimization. In: Advances in Neural Information Processing Systems, pp. 5496–5507 (2019). http://papers.nips.cc/paper/8788-scalable-global-optimization-via-local-bayesian-optimization.pdf

9. Gardner, J., Guo, C., Weinberger, K., Garnett, R., Grosse, R.: Discovering and exploiting additive structure for bayesian optimization. In: Artificial Intelligence and Statistics, pp. 1311–1319. PMLR (2017)
10. Gould, N., Scott, J.: A note on performance profiles for benchmarking software. ACM Transactions on Mathematical Software (TOMS) **43**(2), 1–5 (2016)
11. Gutmann, H.M.: A radial basis function method for global optimization. Journal of global optimization **19**(3), 201–227 (2001)
12. Hansen, N., Ostermeier, A.: Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In: Proceedings of IEEE international conference on evolutionary computation, pp. 312–317. IEEE (1996)
13. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary computation **9**(2), 159–195 (2001)
14. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. Journal of Global optimization **13**(4), 455–492 (1998)
15. Kandasamy, K., Schneider, J., Póczos, B.: High dimensional bayesian optimisation and bandits via additive models. In: International conference on machine learning, pp. 295–304. PMLR (2015)
16. Le Digabel, S.: Algorithm 909: Nomad: Nonlinear optimization with the mads algorithm. ACM Transactions on Mathematical Software (TOMS) **37**(4), 1–15 (2011)
17. Moré, J.J., Wild, S.M.: Benchmarking derivative-free optimization algorithms. SIAM Journal on Optimization **20**(1), 172–191 (2009)
18. Nannicini, G.: On the implementation of a global optimization method for mixed-variable problems. Open Journal of Mathematical Optimization **2**, 1–25 (2021)
19. Powell, M.: The theory of radial basis function approximation in 1990, advances in numerical analysis ii: Wavelets, subdivision, and radial functions (wa light, ed.). Oxford University Press, Oxford **105**, 210 (1992)
20. Regis, R.G.: Stochastic radial basis function algorithms for large-scale optimization involving expensive black-box objective and constraint functions. Computers & Operations Research **38**(5), 837–853 (2011)
21. Regis, R.G., Shoemaker, C.A.: A stochastic radial basis function method for the global optimization of expensive functions. INFORMS Journal on Computing **19**(4), 497–509 (2007)
22. Regis, R.G., Shoemaker, C.A.: Combining radial basis function surrogates and dynamic coordinate search in high-dimensional expensive black-box optimization. Engineering Optimization **45**(5), 529–555 (2013)
23. Wang, J., Clark, S.C., Liu, E., Frazier, P.I.: Parallel bayesian global optimization of expensive functions. arXiv preprint arXiv:1602.05149 (2016)
24. Wang, Z., Gehring, C., Kohli, P., Jegelka, S.: Batched large-scale bayesian optimization in high-dimensional spaces. In: International Conference on Artificial Intelligence and Statistics, pp. 745–754. PMLR (2018)
25. Wu, J., Poloczek, M., Wilson, A.G., Frazier, P.I.: Bayesian optimization with gradients. arXiv preprint arXiv:1703.04389 (2017)
26. Xie, J., Frazier, P.I., Chick, S.E.: Bayesian optimization via simulation with pairwise sampling and correlated prior beliefs. Operations Research **64**(2), 542–559 (2016)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.