MONGOOSE: Path-wise Smooth Bayesian Optimisation via Meta-learning

Anonymous Authors[1]

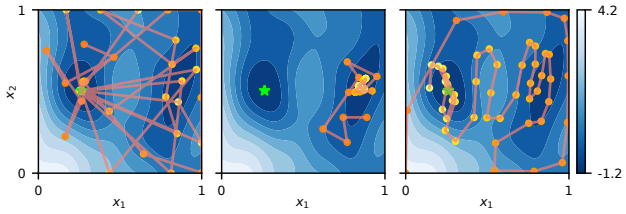
Abstract

In Bayesian optimisation, we often seek to minimise the black-box objective functions that arise in real-world physical systems. A primary contributor to the cost of evaluating such black-box objective functions is often the effort required to prepare the system for measurement. We consider a common scenario where preparation costs grow as the distance between successive evaluations increases. In this setting, smooth optimisation trajectories are preferred and the jumpy paths produced by the standard myopic (i.e. one-step-optimal) Bayesian optimisation methods are sub-optimal. Our algorithm, MONGOOSE, uses a meta-learnt parametric policy to generate smooth optimisation trajectories, achieving performance gains over existing methods when optimising functions with large movement costs.

1. Introduction

The task of optimising high-cost black-box functions is inescapable across science and industry. For many of these problems, evaluating the black-box is expensive, not due to the resources expended to take the measurement itself, but instead due to the substantial *movement cost* required to transition the system to be ready for the next high-quality measurement — a cost that increases with the distance (in the input space) between successive measurements. Example movement costs include: the financial outlay of moving mining machinery between drill sites when seeking areas dense in valuable ores (Jafrasteh & Suárez, 2021); the time taken for mixtures of chemicals to reach steady state when trying to identify optimal mixtures (Teh et al., 2008; Ranković et al., 2022); or the effort required to reconfigure mechanical systems like particle accelerators (Roussel et al., 2021) or heat exchangers (Paleyes et al., 2022).

Bayesian Optimisation (Shahriari et al., 2015, BO) is a popular approach for black-box optimisation under constrained budgets. At first glance, BO appears to be a promising method for the problems above. However, standard BO is not designed for settings with movement cost constraints. As such, most methods, including those driven by acquisition functions such as Expected Improvement, favour reducing uncertainty in previously unexplored areas, a strategy that results in large jumps between successive evaluations. Therefore, while efficient in terms of the number of evaluations, standard BO is not efficient in terms of movement costs (see Figure 1a).



(a) EI (b) EI per unit cost (c) MONGOOSE

Figure 1: 50 minisation steps (orange dots to yellow dots) on a toy function (background). Standard BO with EI (a) incurs large movement costs, whereas EI per unit cost (b) fails to reach the global minima (star). Our non-myopic approach (c) finds the minima whilst following a smooth trajectory.

At the same time, encouraging smooth optimisation paths by simply penalising large movements, e.g. considering the EI per unit movement cost (discussed in (Folch et al., 2022)), can lead to a failure to escape local optima (see Figure 1b). This is due to the myopic nature of such an approach: it takes into account only the immediate benefit provided by making an evaluation. However, in order to acheive a global optimum by following a smooth evaluation path, we must accept the immediate sub-optimality of steadily traversing a low-quality region in order to access new promising areas, instead of jumping to the greedy solution — a trade-off that will never be made under myopia.

Successful movement-cost constrained BO thus requires non-myopic decision making. Unfortunately, there has been limited success in developing non-myopic BO methods. Solving the *multi-step look-ahead problem* (Osborne et al., 2009) is challenging since calculating non-myopic acqui-

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the SPIGM workshop at ICML 2024. Do not distribute.

sition functions requires nested maximisations and expectations when conditioning the surrogate model over each future time step (see (González et al., 2016) for a discussion). Therefore the computational cost of existing non-myopic BO methods like (Jiang et al., 2020b) and (Lee et al., 2021) scales prohibitively for the longer time horizons ($\gg 10$) required for smooth global optimisation.

In this work, we propose a new algorithm, Meta-learning Of Non-myopic Global Optimisation fOr Smooth Exploration (MONGOOSE), for the optimisation of black-box functions under high movement costs (See Figure 1c). We sidestep the need to calculate non-myopic acquisition functions by leveraging recent developments in *memory-based* optimisation to instead learn a non-myopic policy directly. In particular, we train a recurrent neural network to provide efficient cost-efficient optimisation over carefully crafted test functions based on samples from a Gaussian Process (Rasmussen et al., 2006, GP). Our chosen network architecture enjoys an inductive bias for smooth paths and our proposed loss function allows the degree of smoothness to be customised to the task at hand. Finally, we show that MONGOOSE improves over baselines for a variety of test functions.

2. Background

In this work, we seek to find the minimum of a smooth black-box function $f : \mathcal{X} \to \mathbb{R}$ over a compact search space $\mathcal{X} = [0, 1]^d$ under a total evaluation budget of T steps. Critically, we wish to perform this optimisation whilst incurring minimal cumulative moving cost $\mathcal{C}(\tau) = \sum_{t=0}^{T-1} \mathcal{C}(\mathbf{x}_t, \mathbf{x}_{t+1})$. The cost function $\mathcal{C} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ denotes the resources required to move between evaluations at $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$. Our framework is agnostic to the exact form of the cost function, as long as it is differentiable, with the L_1 and L_2 distances being common examples. The remainder of this Section details existing methods that are relevant for optimisation under movement costs, laying out important groundwork for our proposed MONGOOSE algorithm.

2.1. Bayesian Optimisation

In standard Bayesian Optimisation (BO) the goal is typically to minimise f in *as few* evaluations as possible. Although this goal is not guaranteed to correspond to efficient optimisation under movement costs, we introduce it here as BO forms the basis for most existing methods for optimisation under movement costs.

BO achieves high data efficiency by using previously collected function evaluations to build a probabilistic *surrogate model* of the objective function. Typically GPs are used for these surrogates, however neural networks (Snoek et al., 2015) and sparse GPs (Chang et al., 2022; Moss et al., 2023) have also been considered. This surrogate model is then used, through a search strategy known as an *acquisition function* $\alpha : \mathcal{X} \to \mathbb{R}$, to carefully select the next value of $\mathbf{x}$ at which to evaluate f, aiming to focus future resources promising areas of the space. Popular acquisition functions include those based on expected improvement (Jones et al., 1998), knowledge gradient (Frazier et al., 2008), and Thompson sampling (Kandasamy et al., 2018), as well a range of entropy-based methods (Hennig & Schuler, 2012; Hernández-Lobato et al., 2014; Wang & Jegelka, 2017; Moss et al., 2021).

2.2. BO under movement costs

A simple way to adapt BO to provide efficient optimisation with respect to movement costs is to incorporate these movement costs into its acquisition function. For instance the Expected Improvement per unit cost (EIpu) is defined as

$$\alpha_{\text{EIpu}}(\mathbf{x}_t, \mathbf{x}_{t+1}) = \alpha_{\text{EI}}(\mathbf{x}_{t+1})/(\gamma + \mathcal{C}(\mathbf{x}_t, \mathbf{x}_{t+1})), \quad (1)$$

where α_{EI} the standard EI acqusition function and γ is a small tuneable parameter (set as $\gamma = 1$ by (Folch et al., 2022)). Unfortunately, EIpu heavily penalises the acquisition function away from the current location and often struggles to achieve global optimisation due to over-exploitation (recall Figure 1).

The current state-of-the-art BO method for optimisation under movement costs is the Sequential Bayesian Optimisation via Adaptive Connecting Samples (SnAKe) of (Folch et al., 2022). SnaKe follows the shortest path that connects a large number of promising regions, as identified through an approximate Thompson sampling scheme (Wilson et al., 2020; Vakili et al., 2021). However, as demonstrated empirically by (Folch et al., 2022), SnAKe has several shortcomings including the requirement of an additional heuristic to ensure that it avoids getting stuck in local modes and a drop in performance when considering higher dimensions and/or shorter time horizons.

2.3. Memory-Based Optimisation

There is a growing trend of training neural networks as black-box optimisers (Volpp et al., 2019; Lange et al., 2022; Metz et al., 2022; Chen et al., 2022b); that is, teaching a network M_θ to take in t previous evaluations and output a new promising location, i.e. $M_\theta : (\mathcal{X}, \mathcal{Y})^t \to \mathcal{X}$ where θ denotes learnable weights. One immediate advantage over BO-based methods is that generating the next query point requires only a single forward pass of the network rather than the significant expense of fitting a GP and maximising an acquisition function. In particular, as the dimensionality of the problem increases, learning a decision policy directly side-steps the need to optimise an acquisition function in a high dimensional space.

Network Architecture A common choice for meta-optimisers is a memory-based network (e.g. recurrent neural networks) (Chen et al., 2022a), which typically stores an internal memory state that summarises the history of observations $\{(\mathbf{x}_{t'}, f(\mathbf{x}_{t'})\}_{t'=1}^{t}$ and merges it with a current observation $(\mathbf{x}_t, f(\mathbf{x}_t)$ to produce a new location at which to evaluate $\mathbf{x}_{t+1}$. Such meta-trained meomory-based optimisers can memorise an effective adaptive search strategy based on the information learnt during meta-training, and reassuringly, they are known to achieve close to (Bayes) optimal performance (Ortega et al., 2019; Mikulik et al., 2020). Consequently, a widely used architecture for memory-based optimisers is the Long Short-Term Memory (LSTM) of (Hochreiter & Schmidhuber, 1997) (see for example (Chen et al., 2017; Mikulik et al., 2020; Chen et al., 2022a; Ni et al., 2021) or (Metz et al., 2022)).

Training Objective To train a *memory-based optimiser* it is common to use a meta-learning approach. More precisely, the network is trained to optimise a large set of objectives drawn from a distribution over functions which hopefully captures the true target objective, e.g. (Chen et al., 2017) use functions sampled from a Gaussian process prior.

When measuring the performance of a particular optimiser over a fixed optimisation budget T, a natural non-myopic metric is to consider the overall improvement found by the optimiser. More precisely, we can write this training objective as

$$\mathcal{L}(\theta) = \mathbb{E}_f \left[f(\mathbf{x}_1) - \min_{t=1,\dots,T} f(\mathbf{x}_t) \right], \tag{2}$$

where the expectation is taken with respect to a chosen prior over training functions $p(f)$. $\mathbf{x}_t$ denotes the location of the t^{th} evaluation chosen by our optimiser when applied to the function f, so $\mathbf{x}_t$ is a function of both θ and the previous function evaluations, $f(\mathbf{x}_1), \dots, f(\mathbf{x}_{t-1})$.

Although this training objective (2) appears equivalent to the one discussed by (Chen et al., 2017), due to a subtle implementation detail regarding the "detaching" of gradient terms related to non-myopia, the objective they actually optimise ends up being myopic. In contrast, we do not detach any gradients and instead use the full non-myopic objective for meta-training. Additional discussions and empirical results demonstrating a significant difference in performance between these two approaches are included in Appendix A.2.

Meta-training The objective in Equation 2 is intractable, and therefore we use Monte-Carlo approximations during meta-training. At each optimisation step, we sample a set of B functions $\{f_1, .., f_B\}$ from our prior distribution for f, and roll out our LSTM optimiser for each function, i.e we use the approximate objective

$$\mathcal{L}_{\text{MC}}(\theta) = \frac{1}{B} \sum_{b=1}^{B} (f_b(\mathbf{x}_1) - \min_{t=1,\dots,T} f_b(\mathbf{x}_t)). \tag{3}$$

During meta-training, we *maximise* the objective with respect to the LSTM weights θ using a stochastic optimiser.

If the memory-based optimiser is to be deployed on noisy objective functions, then we can simply add noise to the training functions to account for this, i.e. $f(\mathbf{x}) + \eta(\mathbf{x})$, where η is an arbitrary but known noise distribution. Note that standard BO methods are typically limited to Gaussian noise to ensure computational tractability.

3. MONGOOSE

We now present our proposed algorithm, Meta-learning Of Non-myopic Global Optimisation fOr Smooth Exploration (MONGOOSE), which builds upon recent advances in memory-based meta learning and Bayesian optimisation to provide a black-box function minimiser that is efficient under large movement costs. At a high-level, MONGOOSE follows the ideas of (Chen et al., 2017) and meta-trains an LSTM, M_θ, to optimise black-box functions. However, we introduce a number of key differences, including the use of a full non-myopic objective that incorporates moving cost, a better designed meta-training distribution, and a more efficient sampling and training scheme.

Our proposed MONGOOSE algorithm introduces three extensions to the work of (Chen et al., 2017) which improve the efficiency and applicability of memory-based optimisation. These are

1. A training objective that encourages smooth optimisation paths.
2. A new prior that generates more realistic training objective functions.
3. A light-weight training scheme built upon efficient sampling methods.

We expand on all three of these in the subsequent sections.

3.1. Training Objective for Smooth Paths

We already have a non-myopic training objective for meta-training (2), however, it does not yet favour optimisation paths that incur minimal movement costs. Fortunately, we can easily incorporate a moving cost into our training objective as follows

$$\mathcal{L}_{\text{div}}(\theta) = \frac{\mathcal{L}_{\text{MC}}(\theta)}{1 + \alpha \sum_{1}^{T-1} c(\mathbf{x}_t, \mathbf{x}_{t+1})}. \tag{4}$$

(We also consider an additive moving cost in Appendix A.1.) Here, α is a hyperparameter controlling the weight of moving cost, and $c(\cdot, \cdot)$ is a distance function. Any differentiable function, $c(\mathbf{x}_t, \mathbf{x}_{t+1})$ is admissible.

One of the key advantages of this training objective is that we can control the relative importance of moving costs using α, an important degree of freedom that allows MONGOOSE to be customised to specific problem settings. In contrast, the current state-of-the-art SnAKe (Folch et al., 2022) lacks this flexibility. Figure 2 demonstrates that increasing α trades off cost for exploration, a trade that would be appropriate for problems where movement costs significantly dominate the cost of each function evaluation.

Interestingly, even without any moving penalty (i.e. setting $\alpha = 0$) MONGOOSE still generates relatively smooth trajectories. We suspect this is an inductive-bias of memory-based models, where the memory-state may retain more information from the closest previous evaluation $\mathbf{x}_t$ (see Appendix C for a discussion).

3.2. Injecting Global Structure

To guarantee performance at test time, it is of critical importance that the surrogate objective functions that we minimise at training-time are representative of the true test-time objective function. However, there is an emerging consensus that GP samples may not be representative of real-world objective functions. First, Le Riche & Picheny (2021) and Picheny et al. (2022) emphasise that real-world objectives often have a single global optimum, and "global" structure around that optimum. In contrast, functions sampled from GP priors with e.g. Matérn or squared exponential kernels, have no global structure that extends beyond the GP length-scale, and hence may have many comparably performing minima. Second, (Hvarfner et al., 2022) argue that global minima are likely to lie centrally in the search space (as the search space has been designed by experts to cover the likely value of the global optimum), while, due to the curse of dimensionality, GP samples have their minima focused along the edges of the search domains.

Therefore, to alleviate the shortcomings described above, we deviate from standard training function priors when training MONGOOSE. We sample a quadratic bowl and add this to the training functions sampled from GPs. This addition adds global structure to the training functions and increases the likelihood of having a single central global optima. In particular, to generate a single training objective function, we first generate a sample f from a GP prior, then add a randomly generated convex quadratic,

$$f_{\text{quad}}(\mathbf{x}) = f(\mathbf{x}) + \tfrac{1}{d}(\mathbf{x} - \mathbf{a})^T \mathbf{W}(\mathbf{x} - \mathbf{a}) + c. \quad (5)$$

Here, $\mathbf{W}$ is sampled from a Wishart distribution $\mathcal{W}(\frac{1}{d}\mathbf{I}, d)$ to ensure convexity, $\mathbf{a} \sim U(0.2, 0.8)^d$ to encourage a central minima, and $c = \frac{1}{8d}\sum_{i,j}[\mathbf{W}]_{ij}$ (half the expected maximum value of the quadratic) to ensure that the inclusion of the quadratic doesn't dramatically change the output range of the sampled functions. We found that including this global structure gives an improvement in optimisation performance downstream, especially in higher dimensions (see Appendix B).

3.3. Meta-training By Fourier Features

Recall that calculating our training objective (3) requires the evaluation of K samples from a GP prior, each across T locations. Previous meta-training approaches sample the GP exactly (Chen et al., 2017), however, due to a Cholesky decomposition step (Diggle et al., 1998), this incurs a $O(T^3)$ cost which becomes prohibitively expensive for longer time-horizons.

A natural answer to these scalability issues is to rely instead on an approximate sampling schemes already commonly used throughout BO literature known as Random Fourier Features (RFF). In particular, it is well-known that for many common choices of kernels, GP samples can be expressed as a weighted sum of the kernel's Fourier features (Rahimi & Recht, 2007). This sum can then be truncated to only its M largest contributors, leaving approximate but analytically tractable samples that can be queried with only $O(MT)$ cost. See Appendix A of (Hernández-Lobato et al., 2014) for full details. In our experiments, we found that using these approximate samples for training allowed a dramatic reduction in training costs without a loss in training stability or in the performance of the trained optimisers.

We can now summarise the full algorithm for MONGOOSE in Algorithm 1. Note that the roll-out of MONGOOSE over the B training functions (i.e. line 5 of Algorithm 1) is entirely parallelisable.

Algorithm 1 Training MONGOOSE

Choose Horizon H, # training steps N, Batch size B
$n \in \{1, .., N\}$ Training loop
Generate B approximate GP samples $\{f_1, .., f_B\}$
Add random quadratic effects $f_b \leftarrow f_b + f_{\text{quad}}$
$b \in \{1, .., B\}$ Can be parallelised
$h \in \{1, .., H\}$ Rollout
$\mathbf{x}_h = M_\theta(\{(\mathbf{x}_i, f_b(\mathbf{x}_i))\}_{i=1}^{h-1})$
Use the B roll-outs to calculate $\mathcal{L}_{\text{div}}(\theta)$ Eq. 4
Backpropogate through $\mathcal{L}_{\text{div}}(\theta)$ and update θ
A trained MONGOOSE M_θ

4. Related Work

Cost-constrained BO There are many examples of BO where the cost of evaluations depends on their location (rather than the relative distance from previous evaluations we consider). In this popular setting, building a simple cost-weighted acquisition function like the EI per unit evaluation

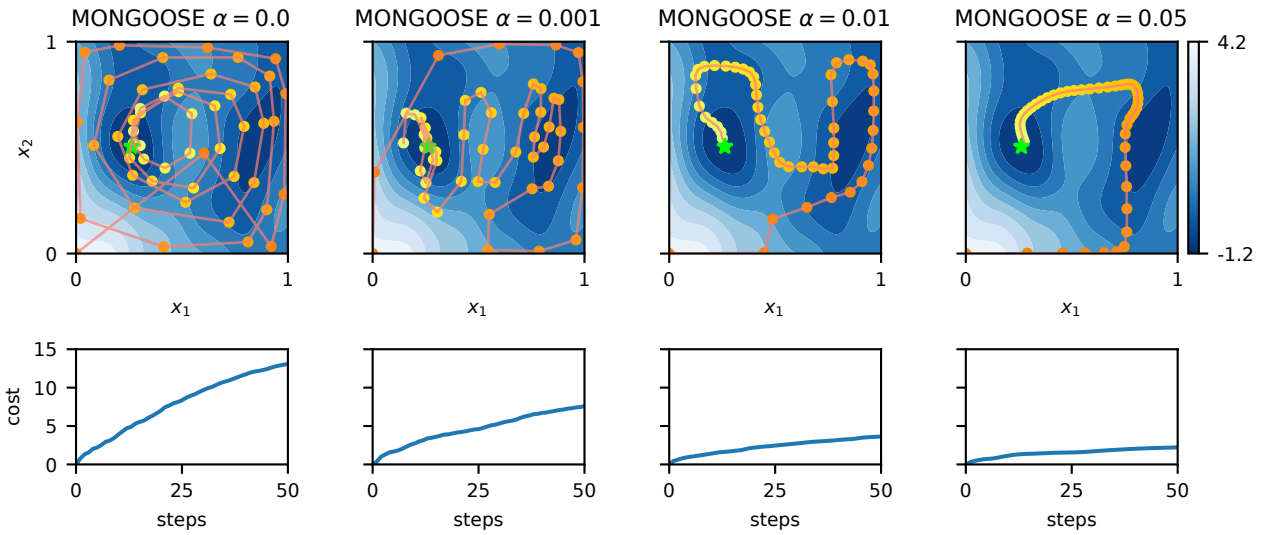


Figure 2: **Top**: trajectories MONGOOSE with different cost scalings on a single function sample from the meta-training distribution (background colour). Cost scalings $\alpha = 0.00, 0.01, 0.05$ from left to right as labelled on titles. Background with colour scale represent the function sample. Orange/yellow dots denote the evaluations chosen by each method, where darker colours (more orange) denote points earlier in the optimisation, and lighter colors (more yellow) denote points later in the optimisation. Consecutive evaluations are joined by lines. **Bottom**: L_2 distance (i.e. moving cost) to traverse each optimisation trajectory.

cost, can sometimes be an effective heuristic, e.g. when tuning the architecture of neural networks where certain design choices increase training times (Snoek et al., 2012) or when multiple evaluation methods are available but each with differing costs, as arise in multi-task (Swersky et al., 2013), multi-source (Poloczek et al., 2017) or multi-fidelity (Moss et al., 2020b) optimisation. Unfortunately, as discussed above and demonstrated in our experiments, applying a simple cost-weighting idea (similar to that proposed by (Roussel et al., 2021)) for the movement cost setting can lead to arbitrarily poor optimisation. Recently, (Lee et al., 2020; 2021) reformulated BO under location dependent costs as a constrained Markov decision process, trading a performance improvement over cost-weighted baselines for significant additional computational complexity. In other related work, Ramesh et al. (2022) considers a similar movement penalty but in a specific contextual BO setting inspired by wind energy systems.

Non-myopic BO When performing global optimisation under a fixed evaluation budget, it should be advantageous to think non-myopically. Consequently, many non-myopic BO approaches have been proposed outside of the cost-constrained setting, ranging from cheap heuristics like GLASSES (González et al., 2016) and BINOCULARS (Jiang et al., 2020a), which approximate multi-step look-ahead as batch experimental design problem, to expensive approximations of optimal non-myopic policies (Jiang et al., 2020b; Yue & Kontar, 2020; Lee et al., 2021) suitable for shorter time-horizons (¡ 10 steps). Note that SnAKE can be interpreted as an extension of GLASSES (González et al., 2016) to the movement constrained setting, achieving a degree of non-myopic decision making via constructing batches.

Meta-learnt optimisers Chen et al. (2017) meta-trained a long short-term memory (LSTM) network (Hochreiter & Schmidhuber, 1997) over samples from a GP. However, their training framework involves conditional sampling of GPs which is both computational and memory intensive. Similarly, (Lange et al., 2022) meta-learned an evolutionary strategy for BO rollout through an attention network, which itself is learned by another outer-loop evolutionary strategy.Volpp et al. (2019) amortised the acquisition function by a meta-learned neural acquisition function over GP a posterior, and subsequently learned a categorical policy on a grid of points through proximal policy optimisation (Schulman et al., 2017). Meta-learning with memory-based agents also achieves state-of-the-art performance in many sequential decision making tasks (Ni et al., 2021). However, none of these existing methods support optimisation under large movement costs.

5. Experiments

We now investigate the performance of MONGOOSE across three different settings: standard BO benchmark functions, across the extensive COCO testing suite(Finck et al., 2010; Hansen et al., 2021), and on a real world example from (Folch et al., 2022). For clarity, all our results follow a similar format, presenting regret against the movement costs incurred over 50 (main text) and 100 (Appendix E) evaluations. All results are based on 50 runs across different random seeds except for MONGOOSE which, due to computational considerations, was ran 10 times for each experiment. Results on noiseless functions are included in the main text. See Appendix D for the corresponding results on noisy objective functions.

5.1. Implementation Details

MONGOOSE Our architecture comprises an LSTM with a hidden cell state dimension of 128, and a decoder with a sigmoid activation that maps hidden states to locations in the search space, with some additional design choices that were helpful in improving training stability. Firstly, we initialised MONGOOSE with a single evaluation at the origin $\mathbf{x}_0 = \mathbf{0}$ (i.e a corner of the search space), with all subsequent evaluations chosen by the model. We found that starting with a randomly located evaluation could lead to less stable model fitting. Secondly, we also found curriculum learning (Bengio et al., 2009) to be important for stability, i.e. we began the optimisation process with shorter horizon lengths and gradually increased it to the desired longer horizons. Each curriculum phase comprised $5,000$ optimisation steps, with each training loss evaluation calculated using a new random batch of 128 training functions. Back propagation through time is used to collect gradients (Werbos, 1990). Finally, we decayed the learning rate from $1\text{e}{-3}$ to $1\text{e}{-4}$ when the curriculum's horizon length reaches 40.

To generate each function used to train MONGOOSE, we first sample a per-dimension lengthscale vector $\ell \in \mathbb{R}^d$ from an inverse Gamma distribution with 99% confidence interval at $[0.1, 0.4]$, and then use this length-scale to build a GP with a Màtern 5/2 kernel with unit variance from which we approximately sample using 100 RFFs. Our choice of randomly sampled lengthscale gives the GP sample variability across input dimensions while being realistic and covers a wide range of possible test functions. The source code for our experiments has been made publicly available[1].

Competitors We use the implementation for EI, EIpu and SnAKe provided by Folch et al. (2022)[2] based on the BOTorch BO library (Balandat et al., 2020). We follow the recommendations of (Folch et al., 2022), setting SnAKe's ϵ-Point deletion scale to $\epsilon = 0.1$ (a tune-able parameter that helps encourage global exploration) and EIpu's cost-scale coefficient to $\gamma = 1$ (for other choices of γ, see Appendix F).

Critical to the performance of BO methods, is access to an initial set of evaluations, from which reliable estimates of model parameters (e.g. lengthscales) can be calculated. Under movement costs, standard space-filling designs incur significant costs and so are likely sub-optimal, however, reliable estimates of model parameters are still required to ensure effective optimisation. We follow the setup of (Folch et al., 2022) and "warm-start" the BO methods (SnAKE, EI and EIpu) by providing them with a reasonable initialisation of GP model parameters (as calculated over an initial design of $10d$ points). As these evaluations are not used directly to fit surrogate models (only indirectly to provide an initial lengthscale), (Folch et al., 2022) chose not to include the cost of this design in the reported cost of their algorithm, a convention we also follow. In contrast, MONGOOSE starts from scratch from a single evaluation at the origin, i.e. with no warm-starting. Despite this substantial advantage given to the baseline methods, we will see that MONGOOSE still achieves superior performance.

5.2. Bayesian optimisation benchmarks

Firstly, we investigate the performance of MONGOOSE on standard BO benchmark functions as presented in Fig. 3. In lower dimensions all algorithms perform similarly, however, when considering higher dimensions (> 3), MONGOOSE consistently achieves lower regret with lower cost, a difference especially pronounced on the challenging highly multi-modal Ackley function. Note that these results match those claimed for EI, EIpu and SnAKe in Figure 11(b), Figure 12(b), Figure 13(b), Figure 14(b), and Figure 15(b) of Folch et al. (2022).

5.3. COCO test suite

For a more thorough evaluation across different types of functions and across dimensions, we now consider the challenging COCO (COmparing Continuous Optimisers) test suite (Finck et al., 2010; Hansen et al., 2021), a suite of 23 functions designed to benchmark black-box optimisers. Each function designed specifically to exhibit different attributes (e.g. multi-modality, low/high conditioning, weak/adequate global structure) and can be defined for arbitrary dimensions. We standardised these functions to make their values lie in a reasonable range (see Appendix G for more details). The amortised results for dimensions two to six are included in Figure 4 (see Appendix H for a per-function breakdown). MONGOOSE reliably achieves the best tradeoff between movement costs and regret across across all dimensions. We believe that the poor performance

[1] https://anonymous.4open.science/r/mongoose_submission-5131/

[2] https://github.com/cog-imperial/SnAKe

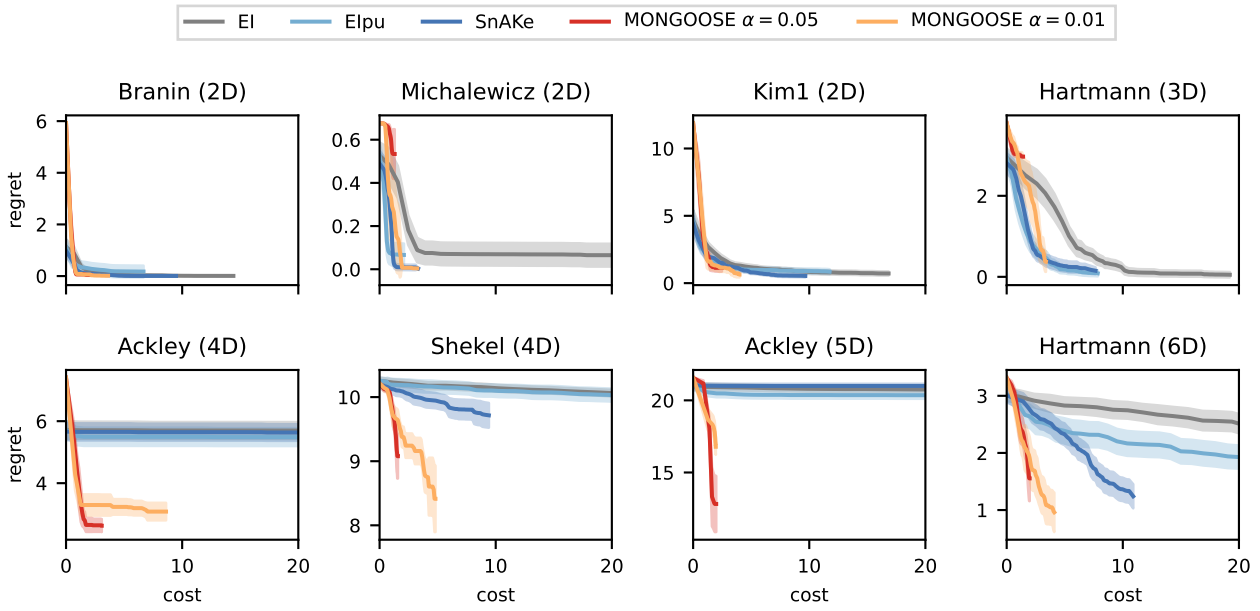


Figure 3: Regret versus cost on standard benchmark objective functions for two versions of MONGOOSE and BO baselines. We plot the mean and a 90% confidence interval of regret for each method.

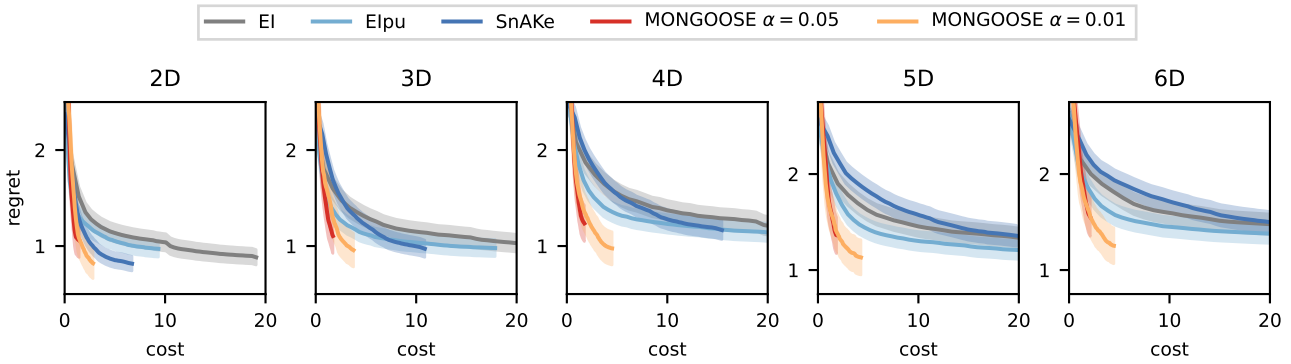


Figure 4: Regret against cost averaged across 24 Coco functions for a range of dimensions.

of SnAKe in five and size dimensions is due to the requirement for a batch of points to achieve good coverage of the space, which becomes increasingly difficult in higher dimensions and under time horizons of only 50. In Appendix E, we show similar results for time horizons of 100 evaluations.

The computational overhead incurred by MONGOOSE when optimising the COCO functions is around three orders of magnitude faster than achieved by the BO baseline methods (see Table 1). Of course, MONGOOSE has the additional cost of requiring meta-training, however, as this takes less than 30 minutes on one RTX2080Ti (when considering a 50 step time horizon) and only needs to be performed once for each considered input dimensionality (i.e. not for each objective function), we do not consider meta-training a serious computational bottleneck.

Table 1: Averaged time for 50 optimisation steps (in seconds) over the COCO test suite.

methods	2D	3D	4D	5D	6D
EI	33	33	35	36	36
EIpu	33	34	36	36	37
SnAKe	23	33	46	70	92
MONGOOSE	**0.02**	**0.02**	**0.02**	**0.02**	**0.02**

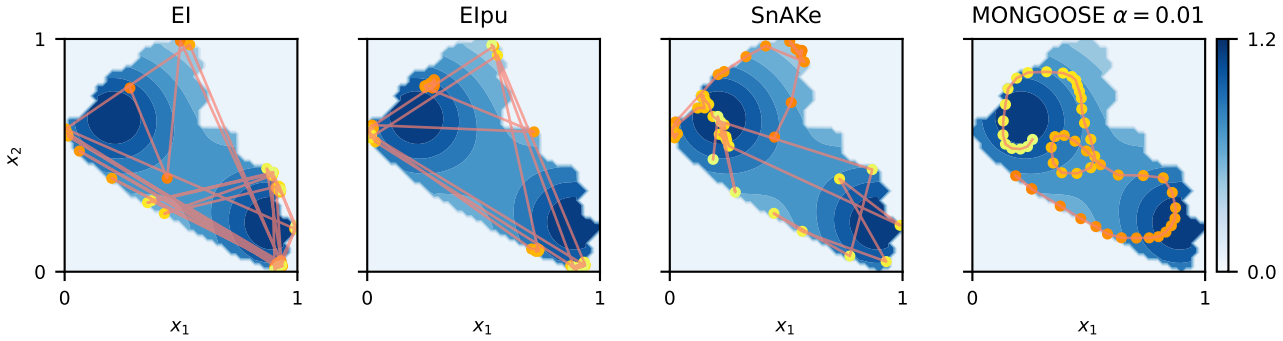


Figure 5: Optimisation trajectories generated when searching for contaminates across the Ypacarai Lake.

5.4. Real world example

For our final example, we turn to the Ypacarai Lake problem (Samaniego et al., 2021; Folch et al., 2022) — a real world black-box optimisation problem that suffers from substantial movement costs. Here, the task is to direct an autonomous surface vehicle to locate contamination sources in the lake, thus travelling a minimal distance is preferred to minimise time and energy consumption. The ground-truth contamination levels over the lake are given over a fine grid. For the BO baselines of EI, EIpu and SnAKe, we use this pre-specified grid as their search space, whereas for MONGOOSE, we project the locations to closest grid point and evaluate the objective at the projected location. Figure 5 compares the trajectories from a single run of EI, EIpu, SnAKe and MONGOOSE, demonstrating that MONGOOSE with $\alpha = 0.01$ is able to generate an entirely smooth trajectory that explores both modes. Figure 6 shows the maximum contamination found against distance travelled by different methods.

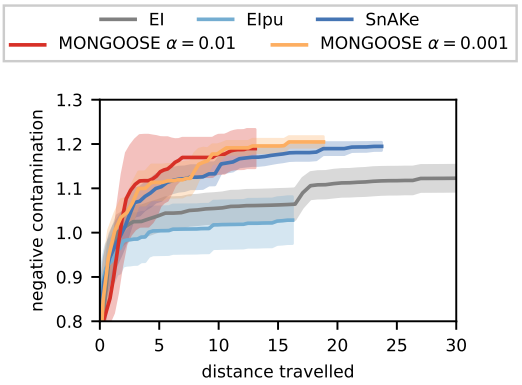


Figure 6: The maximum contamination found against distance travelled by EI, EIpu, SnAKe and MONGOOSE.

6. Discussion

In this work, we developed a memory-based meta-learning approach for the optimisation of black box functions where inputs incur large costs. and our results showed MONGOOSE performs better than competing methods (EI, EIpu and SnAKe) over horizons of 50-100 steps, especially in higher dimensions.

In future work we will investigate the use of dimensional agnostic architectures to avoid the need to train separate network for objective functions with different input dimensions. Attention-based architectures (Lee et al., 2019; Simpson et al., 2021) may provide a solution, with the additional benefit of being invariant to the ordering of query points (a property that should hold for Bayes optimal agents, see (Ortega et al., 2019) or (Mikulik et al., 2020)). Another open question is how to extend memory-based optimisers to non-Euclidean search spaces, a jump recently made by BO in the context of gene design (Moss et al., 2020a), molecular search (Griffiths et al., 2022a;b) and combinatorial optimisation (Deshwal et al., 2021).

References

Balandat, M., Karrer, B., Jiang, D., Daulton, S., Letham, B., Wilson, A. G., and Bakshy, E. Botorch: A framework for efficient monte-carlo bayesian optimization. *NeurIPS*, 2020.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. Curriculum learning. In *ICML*, 2009.

Chang, P. E., Verma, P., John, S., Picheny, V., Moss, H., and Solin, A. Fantasizing with dual gps in bayesian optimization and active learning. *arXiv*, 2022.

Chen, T., Chen, X., Chen, W., Heaton, H., Liu, J., Wang, Z., and Yin, W. Learning to optimize: A primer and a benchmark. *JMLR*, 2022a.

Chen, X., Chen, T., Cheng, Y., Chen, W., Awadallah, A. H., and Wang, Z. Scalable learning to optimize: A learned

optimizer can train big models. In *European Conference on Computer Vision (ECCV 2022)*, 2022b.

Chen, Y., Hoffman, M. W., Colmenarejo, S. G., Denil, M., Lillicrap, T. P., Botvinick, M., and Freitas, N. Learning to learn without gradient descent by gradient descent. In *ICML*, 2017.

Deshwal, A., Belakaria, S., and Doppa, J. R. Mercer features for efficient combinatorial bayesian optimization. In *AAAI*, 2021.

Diggle, P. J., Tawn, J. A., and Moyeed, R. A. Model-based geostatistics. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 1998.

Finck, S., Hansen, N., Ros, R., and Auger, A. Real-parameter black-box optimization benchmarking 2009: Presentation of the noiseless functions. Technical report, Citeseer, 2010.

Folch, J. P., Zhang, S., Lee, R. M., Shafei, B., Walz, D., Tsay, C., van der Wilk, M., and Misener, R. Snake: Bayesian optimization with pathwise exploration. In *NeurIPS*, 2022.

Frazier, P. I., Powell, W. B., and Dayanik, S. A knowledge-gradient policy for sequential information collection. *SICON*, 2008.

González, J., Osborne, M., and Lawrence, N. Glasses: Relieving the myopia of bayesian optimisation. In *AISTATS*, 2016.

Griffiths, R.-R., Greenfield, J. L., Thawani, A. R., Jamasb, A. R., Moss, H. B., Bourached, A., Jones, P., McCorkindale, W., Aldrick, A. A., Fuchter, M. J., et al. Data-driven discovery of molecular photoswitches with multioutput gaussian processes. *Chemical Science*, 2022a.

Griffiths, R.-R., Klarner, L., Moss, H. B., Ravuri, A., Truong, S., Rankovic, B., Du, Y., Jamasb, A., Schwartz, J., Tripp, A., et al. Gauche: A library for gaussian processes in chemistry. *arXiv*, 2022b.

Hansen, N., Auger, A., Ros, R., Mersmann, O., Tušar, T., and Brockhoff, D. Coco: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 2021.

Hennig, P. and Schuler, C. J. Entropy search for information-efficient global optimization. *JMLR*, 2012.

Hernández-Lobato, J. M., Hoffman, M. W., and Ghahramani, Z. Predictive entropy search for efficient global optimization of black-box functions. *arXiv*, 2014.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 1997.

Hvarfner, C., Stoll, D., Souza, A., Lindauer, M., Hutter, F., and Nardi, L. π bo: Augmenting acquisition functions with user beliefs for bayesian optimization. *arXiv*, 2022.

Jafrasteh, B. and Suárez, A. Objective functions from bayesian optimization to locate additional drillholes. *Computers & Geosciences*, 2021.

Jiang, S., Chai, H., Gonzalez, J., and Garnett, R. Binoculars for efficient, nonmyopic sequential experimental design. In *ICML*, 2020a.

Jiang, S., Jiang, D., Balandat, M., Karrer, B., Gardner, J., and Garnett, R. Efficient nonmyopic bayesian optimization via one-shot multi-step trees. *NeurIPS*, 2020b.

Jones, D. R., Schonlau, M., and Welch, W. J. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 1998.

Kandasamy, K., Krishnamurthy, A., Schneider, J., and Póczos, B. Parallelised Bayesian optimisation via Thompson sampling. In *AISTATS*, 2018.

Lange, R. T., Schaul, T., Chen, Y., Zahavy, T., Dallibard, V., Lu, C., Singh, S., and Flennerhag, S. Discovering evolution strategies via meta-black-box optimization. *arXiv*, 2022.

Le Riche, R. and Picheny, V. Revisiting bayesian optimization in the light of the coco benchmark. *Structural and Multidisciplinary Optimization*, 2021.

Lee, E., Eriksson, D., Bindel, D., Cheng, B., and Mccourt, M. Efficient rollout strategies for bayesian optimization. In *UAI*, 2020.

Lee, E. H., Eriksson, D., Perrone, V., and Seeger, M. A nonmyopic approach to cost-constrained bayesian optimization. In *UAI*, 2021.

Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., and Teh, Y. W. Set transformer: A framework for attention-based permutation-invariant neural networks. In *ICML*, 2019.

Metz, L., Harrison, J., Freeman, C. D., Merchant, A., Beyer, L., Bradbury, J., Agrawal, N., Poole, B., Mordatch, I., Roberts, A., et al. Velo: Training versatile learned optimizers by scaling up. *arXiv*, 2022.

Mikulik, V., Delétang, G., McGrath, T., Genewein, T., Martic, M., Legg, S., and Ortega, P. Meta-trained agents implement bayes-optimal agents. *NeurIPS*, 2020.

Moss, H., Leslie, D., Beck, D., Gonzalez, J., and Rayson, P. Boss: Bayesian optimization over string spaces. *NeurIPS*, 2020a.

Moss, H. B., Leslie, D. S., and Rayson, P. Mumbo: Multi-task max-value bayesian optimization. In *Machine Learning and Knowledge Discovery in Databases*, 2020b.

Moss, H. B., Leslie, D. S., Gonzalez, J., and Rayson, P. Gibbon: General-purpose information-based bayesian optimisation. *JMLR*, 2021.

Moss, H. B., Ober, S. W., and Picheny, V. Inducing point allocation for sparse gaussian processes in high-throughput bayesian optimisation. *arXiv*, 2023.

Ni, T., Eysenbach, B., and Salakhutdinov, R. Recurrent model-free rl is a strong baseline for many pomdps. *arXiv*, 2021.

Ortega, P. A., Wang, J. X., Rowland, M., Genewein, T., Kurth-Nelson, Z., Pascanu, R., Heess, N., Veness, J., Pritzel, A., Sprechmann, P., et al. Meta-learning of sequential strategies. *arXiv*, 2019.

Osborne, M. A., Garnett, R., and Roberts, S. J. Gaussian processes for global optimization. In *LION*, 2009.

Paleyes, A., Moss, H. B., Picheny, V., Zulawski, P., and Newman, F. A penalisation method for batch multi-objective bayesian optimisation with application in heat exchanger design. *arXiv*, 2022.

Picheny, V., Moss, H., Torossian, L., and Durrande, N. Bayesian quantile and expectile optimisation. In *UAI*, 2022.

Poloczek, M., Wang, J., and Frazier, P. Multi-information source optimization. *NeurIPS*, 2017.

Rahimi, A. and Recht, B. Random features for large-scale kernel machines. *NeurIPS*, 2007.

Ramesh, S. S., Sessa, P. G., Krause, A., and Bogunovic, I. Movement penalized bayesian optimization with application to wind energy systems. *NeurIPS*, 2022.

Ranković, B., Griffiths, R.-R., Moss, H. B., and Schwaller, P. Bayesian optimisation for additive screening and yield improvements in chemical reactions–beyond one-hot encodings. *arXiv*, 2022.

Rasmussen, C. E., Williams, C. K., et al. *Gaussian processes for machine learning*. Springer, 2006.

Roussel, R., Hanuka, A., and Edelen, A. Multiobjective bayesian optimization for online accelerator tuning. *Physical Review Accelerators and Beams*, 2021.

Samaniego, F. P., Reina, D. G., Marín, S. L. T., Arzamendia, M., and Gregor, D. O. A bayesian optimization approach for water resources monitoring through an autonomous surface vehicle: The ypacarai lake case study. *IEEE Access*, 2021.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv*, 2017.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and De Freitas, N. Taking the human out of the loop: A review of Bayesian optimization. *IEEE*, 2015.

Simpson, F., Davies, I., Lalchand, V., Vullo, A., Durrande, N., and Rasmussen, C. E. Kernel identification through transformers. *NeurIPS*, 2021.

Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms. *NeurIPS*, 2012.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., and Adams, R. Scalable bayesian optimization using deep neural networks. In *ICML*, 2015.

Swersky, K., Snoek, J., and Adams, R. P. Multi-task bayesian optimization. *NeurIPS*, 2013.

Teh, S.-Y., Lin, R., Hung, L.-H., and Lee, A. P. Droplet microfluidics. *Lab on a Chip*, 2008.

Vakili, S., Moss, H., Artemev, A., Dutordoir, V., and Picheny, V. Scalable thompson sampling using sparse gaussian process models. *NeurIPS*, 2021.

Volpp, M., Fröhlich, L. P., Fischer, K., Doerr, A., Falkner, S., Hutter, F., and Daniel, C. Meta-learning acquisition functions for transfer learning in bayesian optimization. *arXiv*, 2019.

Wang, Z. and Jegelka, S. Max-value entropy search for efficient Bayesian optimization. In *ICML*, 2017.

Werbos, P. J. Backpropagation through time: what it does and how to do it. *IEEE*, 1990.

Wilson, J., Borovitskiy, V., Terenin, A., Mostowsky, P., and Deisenroth, M. Efficiently sampling functions from gaussian process posteriors. In *ICML*, 2020.

Yue, X. and Kontar, R. A. Why non-myopic bayesian optimization is promising and how far should we look-ahead? a study via rollout. In *AISTATS*, 2020.

A. Alternative objectives

A.1. Additive moving cost

In the main text, we considered incorporating moving cost through division,

$$\mathcal{L}_{\text{div}}(\theta) = \frac{\mathcal{L}(\theta)}{1 + \alpha \sum_t c(\mathbf{x}_t, \mathbf{x}_{t+1})}. \tag{6}$$

Here, we consider the alternative option to add the cost

$$\mathcal{L}_{\text{add}}(\theta) = \mathcal{L}(\theta) + \alpha \sum_{t=1}^{H-1} c(\mathbf{x}_t, \mathbf{x}_{t+1}),, \tag{7}$$

however, this is not the ideal choice for black-box functions since the choice of cost scaling α in the additive case needs to be proportional to the scaling of the function and so is difficult to predetermine. Figure 7 compares their performance on normalised COCO functions, notice that they perform similarly under slightly different choices of cost scaling α. In particular, $\mathcal{L}_{\text{div}}$ with $\alpha = 0.001$ gives very similar performance as $\mathcal{L}_{\text{add}}$ with $\alpha = 0.01$, and $\mathcal{L}_{\text{div}}$ with $\alpha = 0.01$ is similar to $\mathcal{L}_{\text{add}}$ with $\alpha = 0.05$.

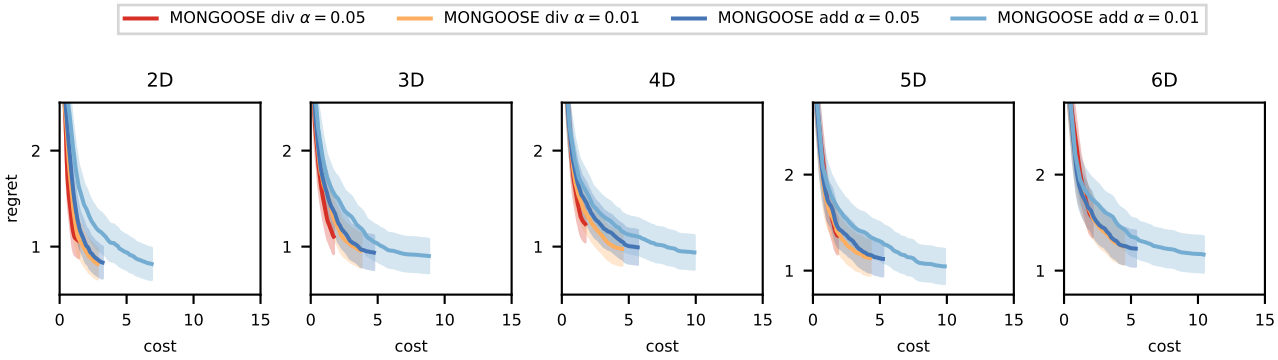


Figure 7: Comparison of $\mathcal{L}_{\text{div}}(\theta)$ versus $\mathcal{L}_{\text{add}}(\theta)$ averaged across the COCO benchmark. Individual plots for each COCO function are shown in Figure 25,26,27,28.

A.2. The myopic objective

The objective we use in our meta-training, as defined in Equation 2, is given by

$$\mathcal{L}(\theta) = \mathbb{E}_f \left[f(\mathbf{x}_1) - \min_{t=1,\dots,T} f(\mathbf{x}_t) \right].$$

This is the expected improvement over our prior with respect to the minimum function value reached during a trajectory of T steps. It is worth noting that this loss can be expressed as a cumulative sum of improvement, which has the same form as the 'observed improvement' proposed by Chen et al. (2017),

$$\mathcal{L}(\theta) = \mathbb{E}_f \left[\sum_{t=1}^{T} \max(\min_{t'=1,\dots,t-1} f(\mathbf{x}_{t'}) - f(\mathbf{x}_t), 0) \right].$$

However, when optimising this loss, Chen et al. (2017) detached the previous best function value $\min_{t'=1,\dots,t} f_k(\mathbf{x}_{t'})$ during back-propagation, effectively making the objective myopic. In contrast, we do not detach gradients and calculate the loss exactly as it is written. To aid intuition, consider a horizon length of 2 with a single training function and $\mathbf{x}_0 = \mathbf{0}$. Under these assumption, our objective becomes

$$\mathcal{L}_{\text{OI}}(\theta) = \max(f(\mathbf{x}_0) - f(\mathbf{x}_1), 0) + \max(\underbrace{\min\{f(\mathbf{x}_0), f(\mathbf{x}_1)\}}_{\text{detach}} - f(\mathbf{x}_2), 0).$$

Now consider detaching the gradient of $f(\mathbf{x}_1)$ from the second term. This leads to myopia because, when updating $\mathbf{x}_1$, its only contribution now comes from the first term which is only a one-step (i.e. myopic) improvement $\max(f(\mathbf{x}_0) - f(\mathbf{x}_1), 0)$. In contrast, a truly non-myopic approach should consider the effect of changing $\mathbf{x}_1$ on all subsequent improvements.

From a more practical perspective, we saw a significant performance degradation when mimicking the gradient detaching of (Chen et al., 2017), as shown in Figure 8.

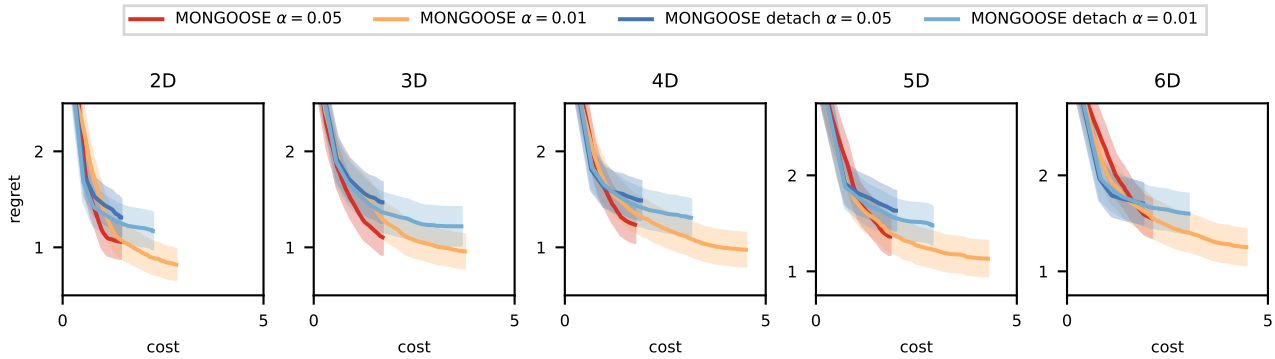


Figure 8: Comparing the non-myopic (red/orange) versus myopic (blues) meta-training objectives averaged across the COCO benchmark, with $\alpha = 0.01, 0.05$. Individual plots for each COCO function are shown in Figure 29,30,31,32.

B. Effect of injecting global structure

We now illustrate the effect of injecting the global structure we described in the main text into GP samples as Figure 9. The major effects of adding global structure include (1) moving global optimum from corners and edges towards the centre and (2) eliminating some modes at corners. Adding this global structure boosts the performance on standard BO benchmarks especially in 4D-6D functions (see Figure 10). When averaged across the COCO benchmark, MONGOOSE trained without global structure consistently achieves lower regret with a lower moving cost, and the advantage grows as dimensionality increases.

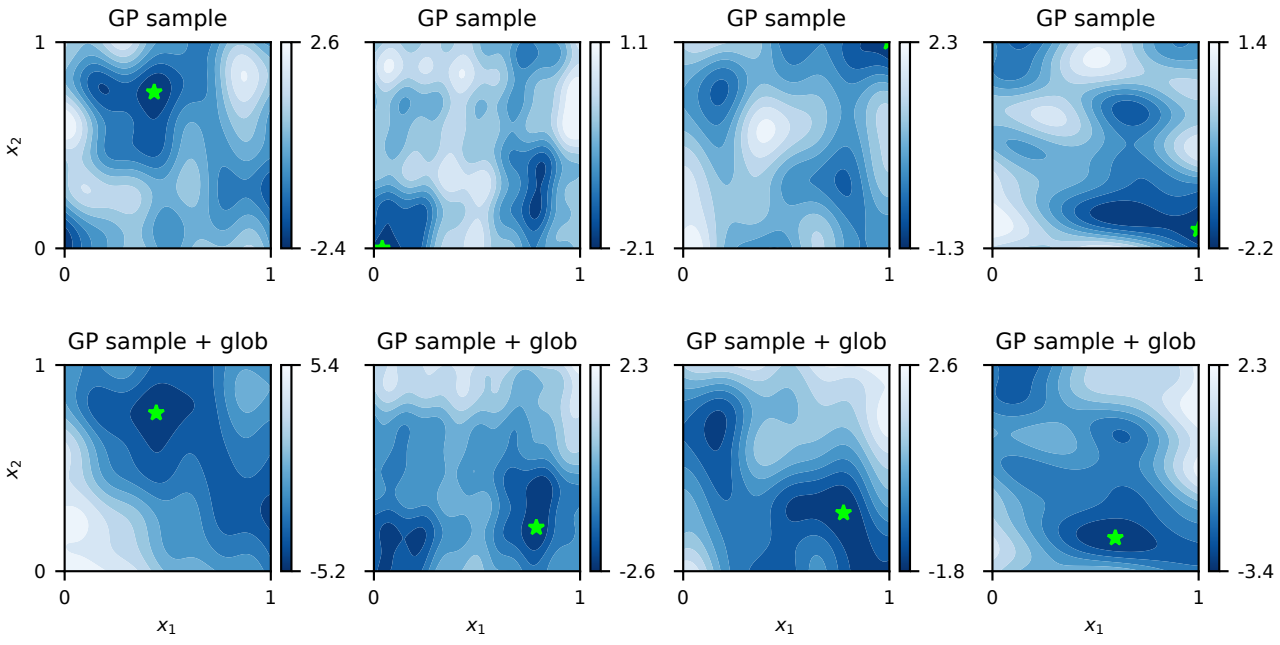


Figure 9: **Top**: original GP samples obtained from a Màtern 5/2 kernel. **Bottom**: The same samples injected with randomly sampled global structure.

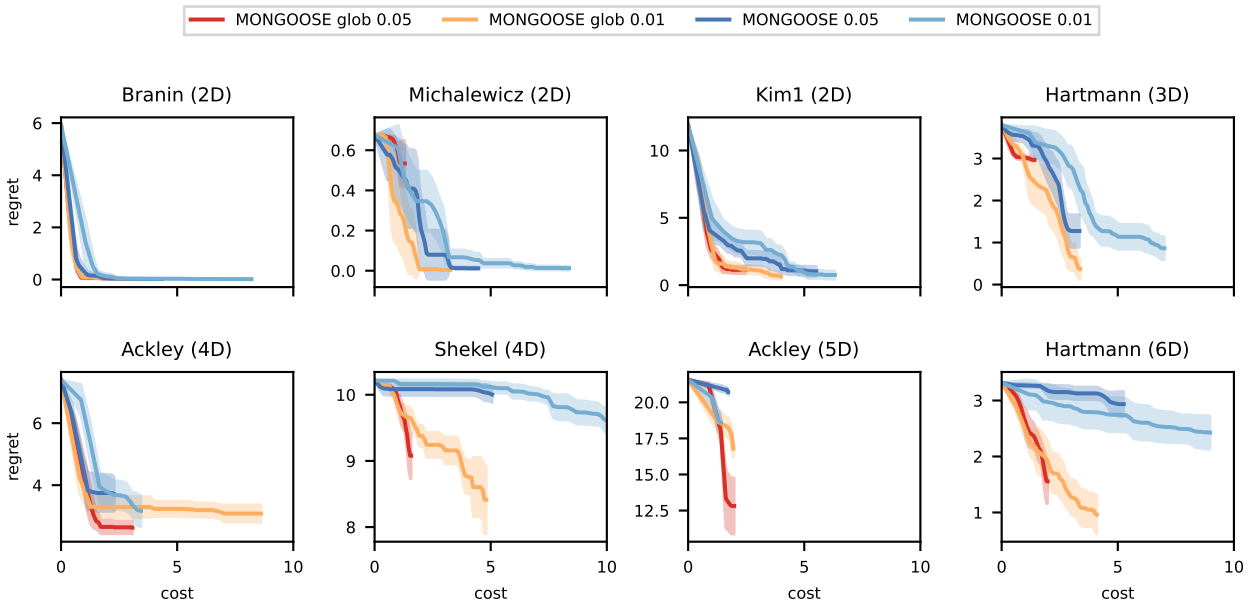


Figure 10: Investigating the effect of adding global structure during meta-training on standard BO benchmarks.

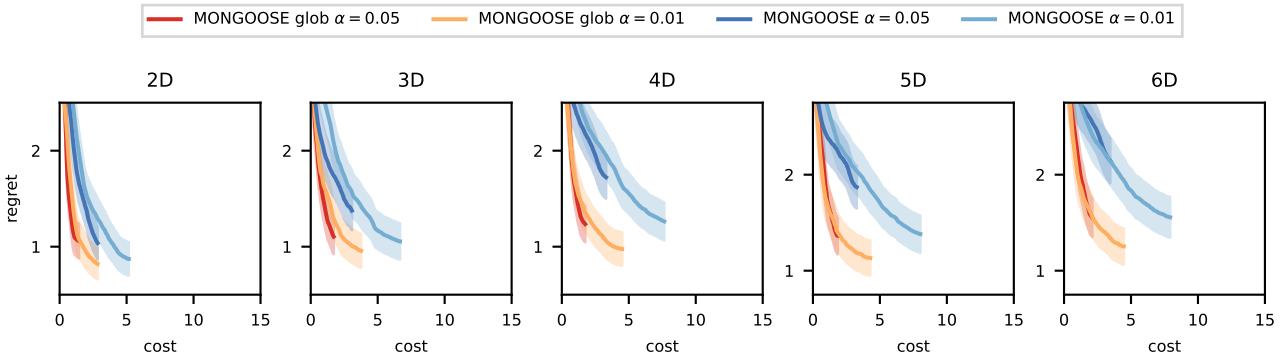
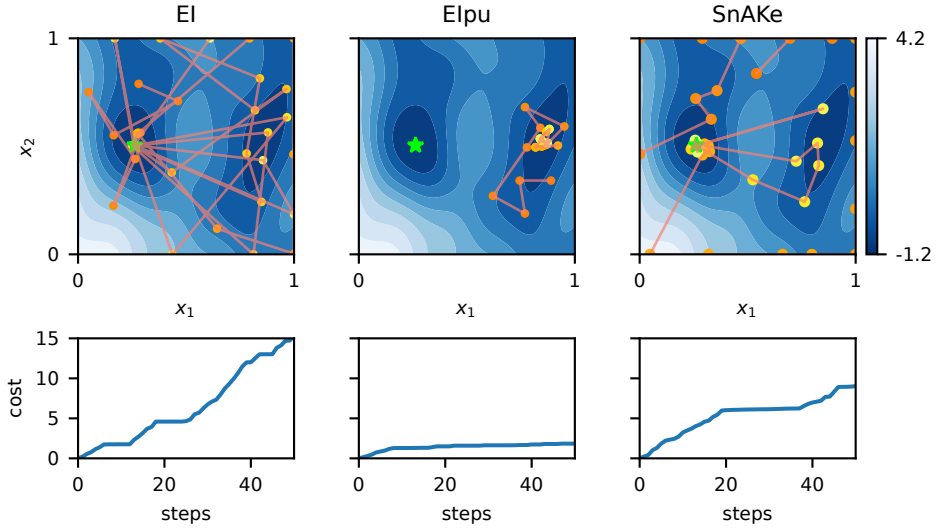


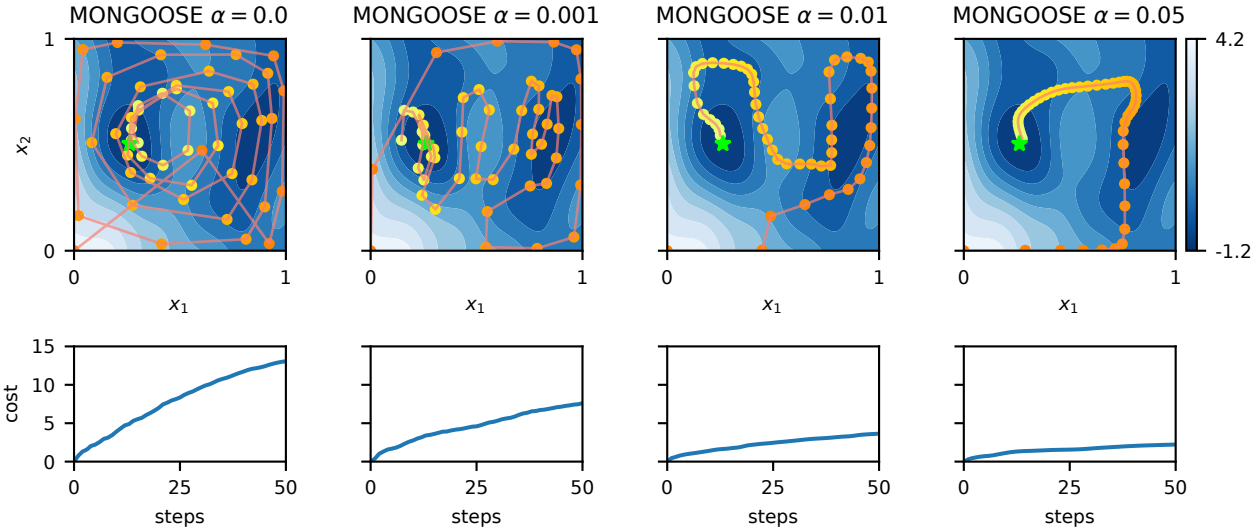
Figure 11: Investigating the effect of adding global structure (red/orange) against standard GP sample (blue) during meta-training averaged across the COCO benchmark, with $\alpha = 0.01, 0.05$. Individual plots for each COCO function are shown in Figure 21,22,23,24.

C. Inductive bias of memory-based meta-optimisers

We found that meta-trained memory-based optimisers using the non-myopic objective (Eq. 2 and Eq. 4) have an inductive bias of generating smooth trajectories with low cost. As illustrated in Figure 12, even with $\alpha = 0$, the trajectory of MONGOOSE is than the jumpy trajectory of EI (Figure 12a). Further evidence is provided in Figure 14, which shows MONGOOSE $\alpha = 0$ outperforming EI, EIpu and SnAKe in terms of averaged regret versus cost on the COCO benchmark (for dimensions higher than 2D). We suspect that MONGOOSE's inducitve bias for smooth paths is due to hidden states in memory-based learners containing more information from closest previous steps, thus biasing the output to lie close to previous outputs.



(a) **Top**: trajectories of EI, EIpu, SnAKe, colour scale same as in Figure 1. **Bottom**: cumulative L_2 cost along the trajectory.



(b) **Top**: trajectories of MONGOOSE with $\alpha = 0, 0.001, 0.01, 0.05$, colour scale same as in Figure 2. **Bottom**: cumulative L_2 cost along the optimisation trajectory.

Figure 12: Comparing trajectories and costs from a single run of EI, EIpu, SnAKe, and MONGOOSE with $\alpha = 0, 0.001, 0.01, 0.05$.

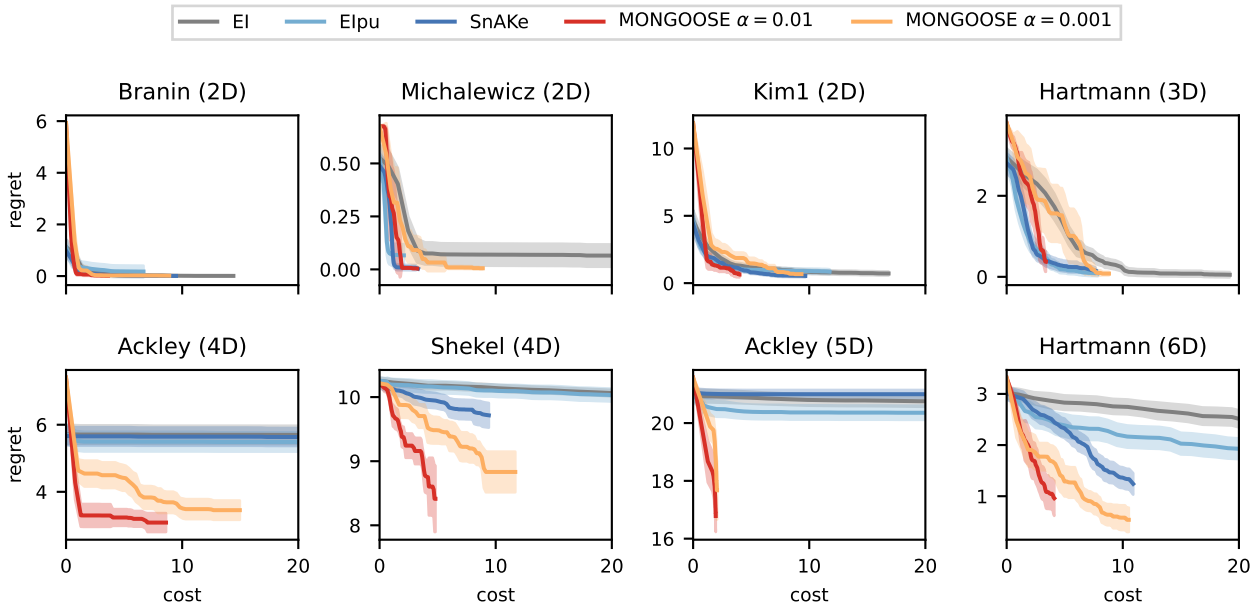


Figure 13: Comparison of EI, EIpu, SnAKe, MONGOOSE with $\alpha = 0.001$ and $\alpha = 0$ on standard BO benchmarks.

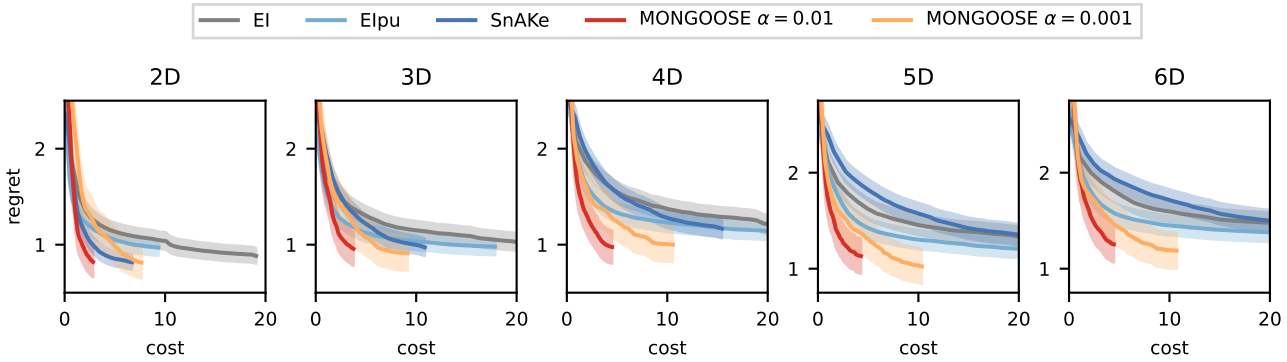


Figure 14: Comparison of EI, EIpu, SnAKe, MONGOOSE with $\alpha = 0.01$ and $\alpha = 0.001$ averaged across the COCO benchmark. Individual plots for each COCO function are shown in Figure 33,34,35,36.

D. Experiments on noisy functions

In the main text, we presented results on noiseless functions. Here, we consider adding Gaussian observation noise to function evaluations. Specifically, we sample noise $\eta \sim \mathcal{N}(0, \sigma^2)$, and let the model (GP for EI, EIpu and SnAKe; LSTM for MONGOOSE) observe new evaluation pair $(\mathbf{x}_t, f(\mathbf{x}_t)+\eta)$, and choose next evaluation location based on noisy observations. when computing the final regret for all methods at test time, we still use the true function value without observation noise. The results for $\sigma^2 = 0.1$ on standard BO benchmarks are shown in Figure 15 and on the COCO benchmark are shown in Figure 16

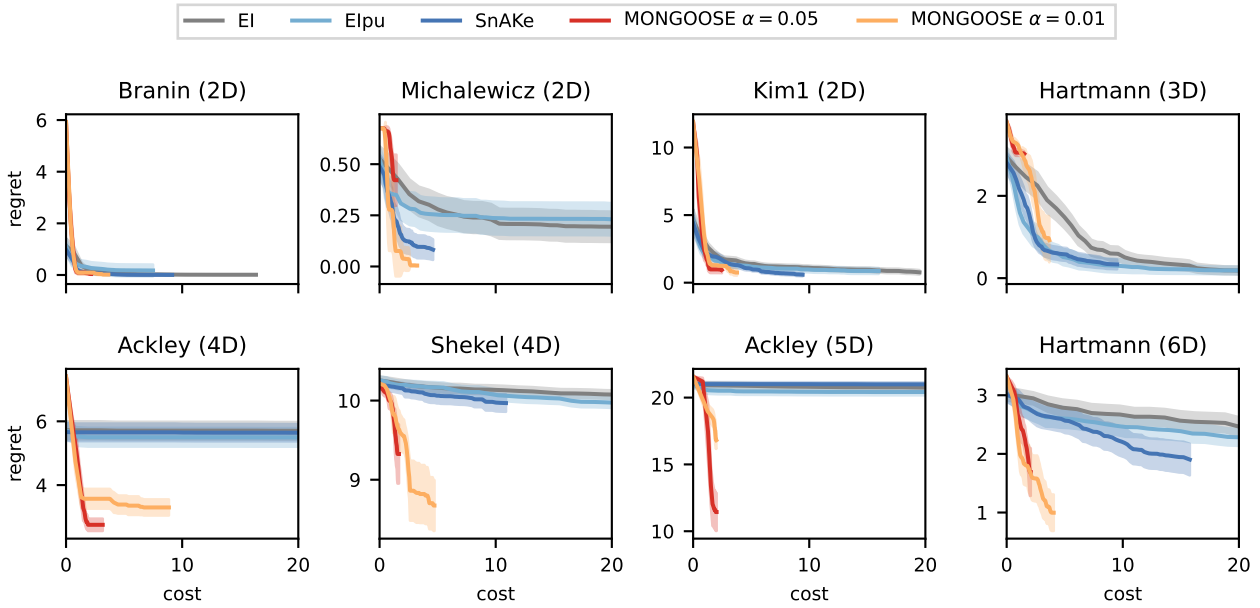


Figure 15: Comparison of EI, EIpu, SnAKe, MONGOOSE with $\alpha = 0.01$ and $\alpha = 0.05$ on standard BO benchmarks with observation noise $\eta \sim \mathcal{N}(0, 0.1)$.

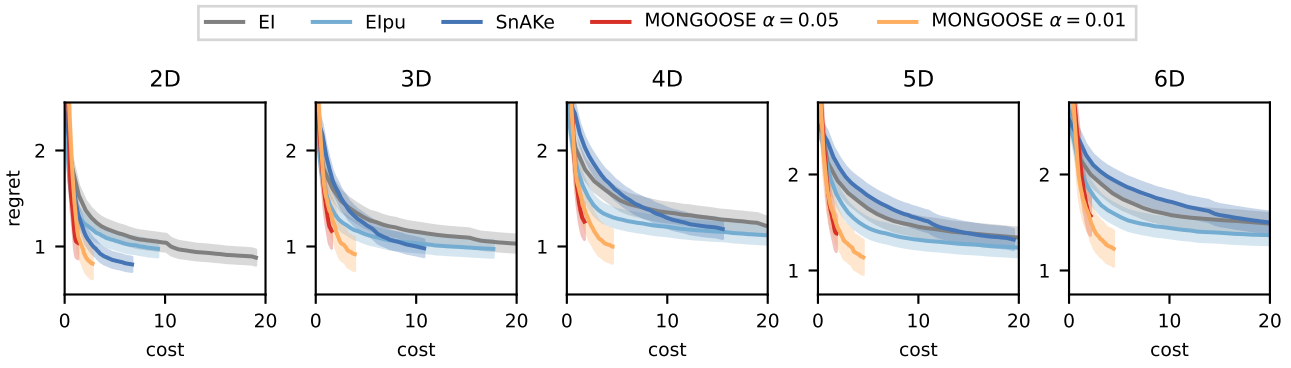


Figure 16: Comparison of EI, EIpu, SnAKe, MONGOOSE with $\alpha = 0.01$ and $\alpha = 0.05$ averaged across the COCO benchmarks with observation noise $\eta \sim \mathcal{N}(0, 0.1)$. Individual plots for each COCO function are shown in Figure 37,38,39,40.

E. Experiments for 100 steps horizon

In the main text, we showed results for a horizon of 50 steps. Here, we show the results for a horizon of 100 steps on standard benchmarks in Figure 17 and on COCO benchmarks Figure 18. Our conclusions from the main paper still hold, although SnAKe does perform noticeable better on Hartmann 6D as well as all COCO functions, which is its expected behaviour as the number of steps in a BO loop grows (Folch et al., 2022).

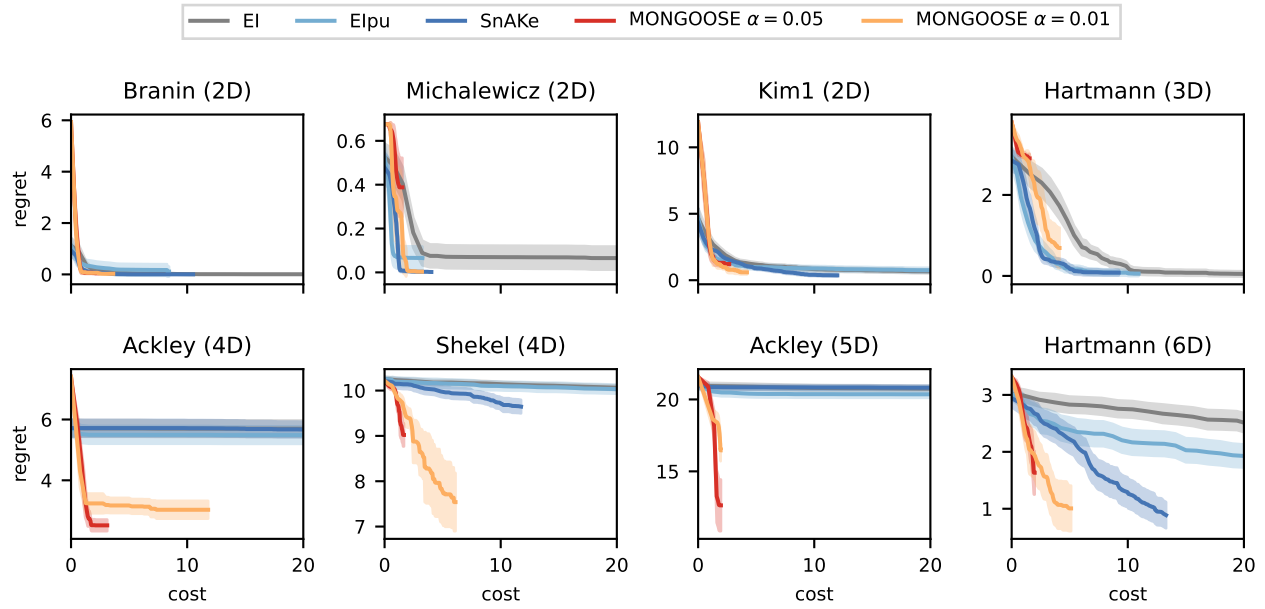


Figure 17: Comparison of EI, EIpu, SnAKe, MONGOOSE with $\alpha = 0.01$ and $\alpha = 0.05$ on standard BO benchmarks for a horizon of 100 steps.

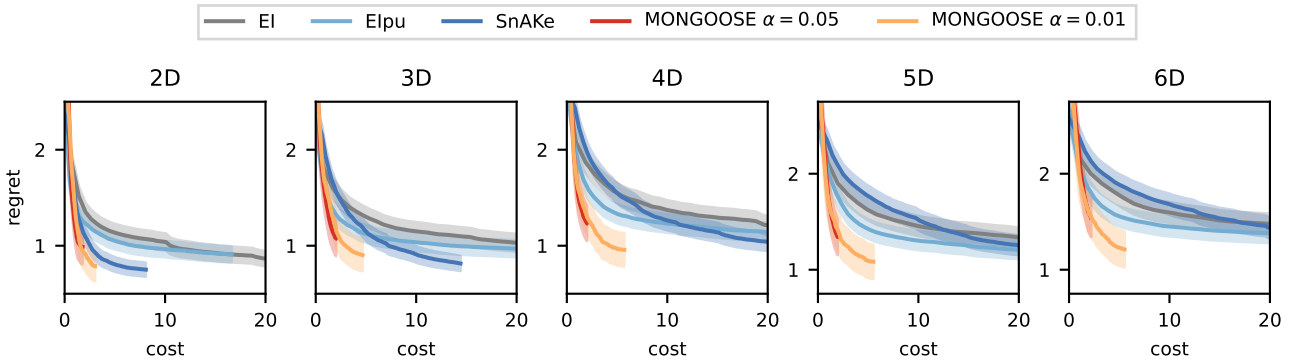


Figure 18: Comparison of EI, EIpu, SnAKe, MONGOOSE with $\alpha = 0.01$ and $\alpha = 0.05$ averaged across the COCO benchmarks for a horizon of 100 steps. Individual plots for each COCO function are shown in Figure 41,42,43,44.

F. EI per unit cost

In this section we investigate the effect of the hyperparameter γ in EI per unit cost (EIpu). Recall EIpu is defined as

$$\text{EIpu}(x) = \frac{\text{EI}(\mathbf{x})}{\gamma + 1}.$$

In the main text, we chose $\gamma = 1$ following (Folch et al., 2022). As demonstrated in Figure 19, EIpu with $\gamma = 1, 0.1, 0.01$ are all outperfomed by MONGOOSE

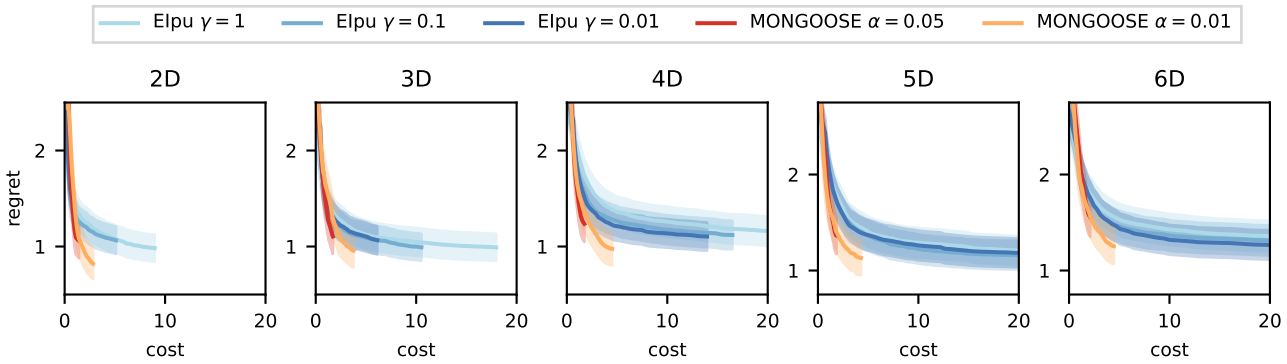


Figure 19: Comparison of EIpu with $\gamma = 0.01, 0.1, 1$, and MONGOOSE with $\alpha = 0.01, 0.05$, averaged across the COCO benchmarks for a horizon of 100 steps. Individual plots for each COCO function are shown in Figure 45,46,47,48.

G. COCO functions

There are a total of 24 functions in the COCO benchmark Finck et al. (2010); Hansen et al. (2021), all of them are positive and have a known global minima with a corresponding minimum function value. Many have random parameters that we can sample to generate slightly different but similar functions. Since not all functions have this randomness and the random parameters are usually just rotations in the input space, we fixed all random parameters for our tests. One potential issue with functions in this benchmark is their outputs have vastly different ranges, for example, the ellipsoidal function (2D) ranges from 0 to 3e7 (Finck et al. (2010, p. 10)), the Rastrigin function (2D) ranges from 0 to 800 (Finck et al. (2010, p. 15)), the (log) Rosenbrock function (2D) ranges from 0 to 4 (Finck et al. (2010, p. 40)), etc. Therefore, we chose to standardise these functions

$$\tilde{f}(\mathbf{x}) = \frac{f(\mathbf{x})}{\max_{\mathbf{x}} f(\mathbf{x})} \times 6 - 3 + f_{\text{opt}},$$

where $\max_{\mathbf{x}} f(\mathbf{x})$ is obtained through random search. Following, (Finck et al., 2010) we add $f_{\text{opt}} \sim U[0, 1]$ for additional randomness of the optimum value. Figure 20 shows plots for all functions after normalisation in the COCO benchmark. Note that since we are plotting with a grid of points, they might not cover the exact minimas/maximas especially when they are in a thin valley, so the minimums/maximums on the colourscales do not represent the exact minimum/maximum values of functions. As described in the main text, for all experiments on the COCO benchmark, we meta-train 10 different MONGOOSE with 10 seeds, and we run EI, EIpu and SnAKe with 50 seeds each.

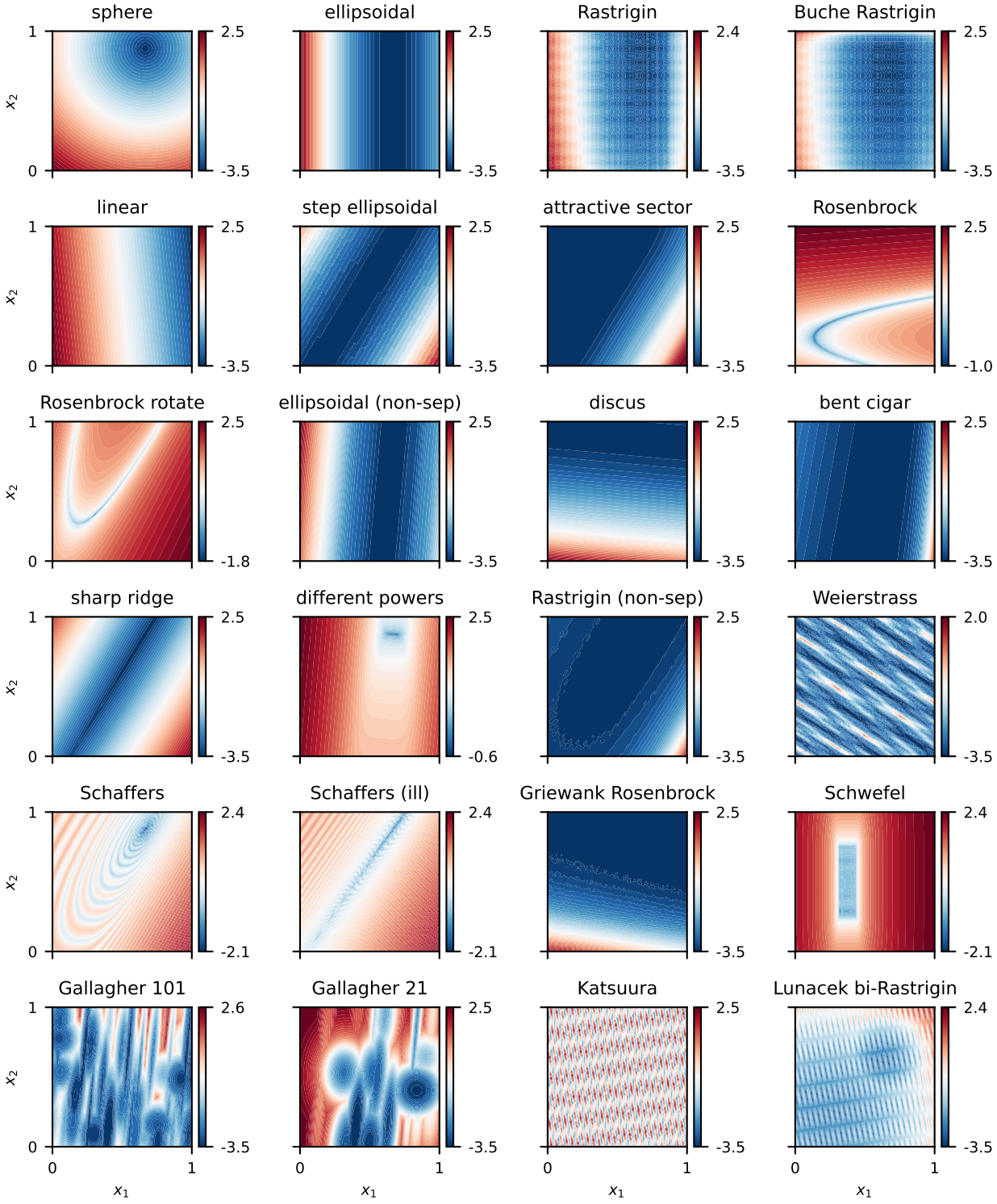


Figure 20: Plots of all 24 functions (noramlised) in the COCO benchmark.

H. COCO individual regret plots

In this final section, we present individual regret plots for each of the 24 COCO functions (Finck et al., 2010; Hansen et al., 2021), which are split into four plots of 6 functions for each setting above with an averaged COCO benchmark plot. We set the same y-axis scale across all regret plot to more easily see the results on which functions contribute more to the averaged differences in regret versus cost.

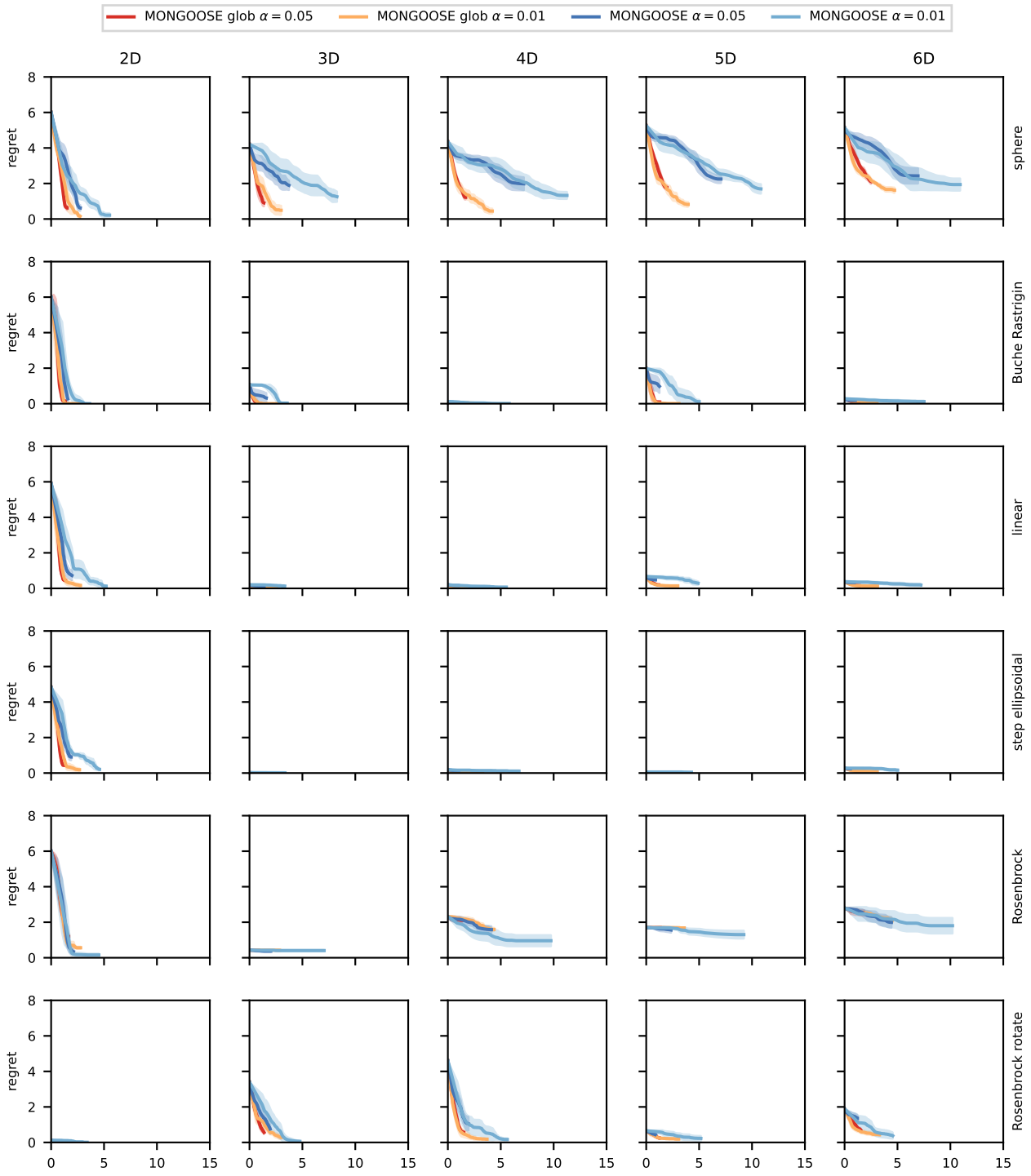


Figure 21: Individual COCO plots for Figure 11. COCO functions 1-6: sphere function, ellipsoidal function, Rastrigin function, Büche-Rastrigin function, linear slope, step ellipsoidal function.

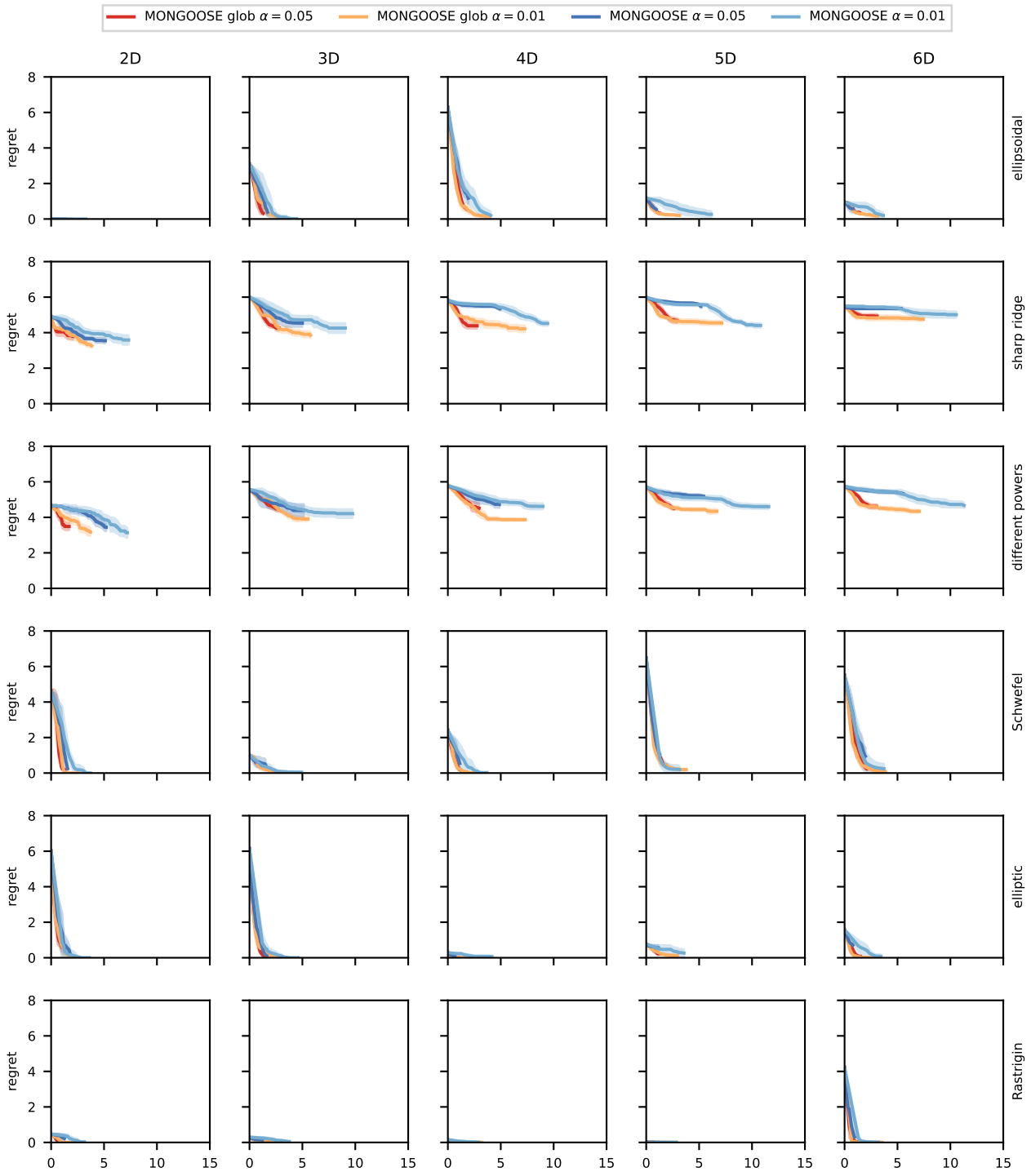


Figure 22: Individual COCO plots for Figure 11. COCO functions 7-12: attractive sector function, Rosenbrock (original) function, Rosenbrock (rotated) function, ellipsoidal (non-separable) function, discus function, bent cigar function.

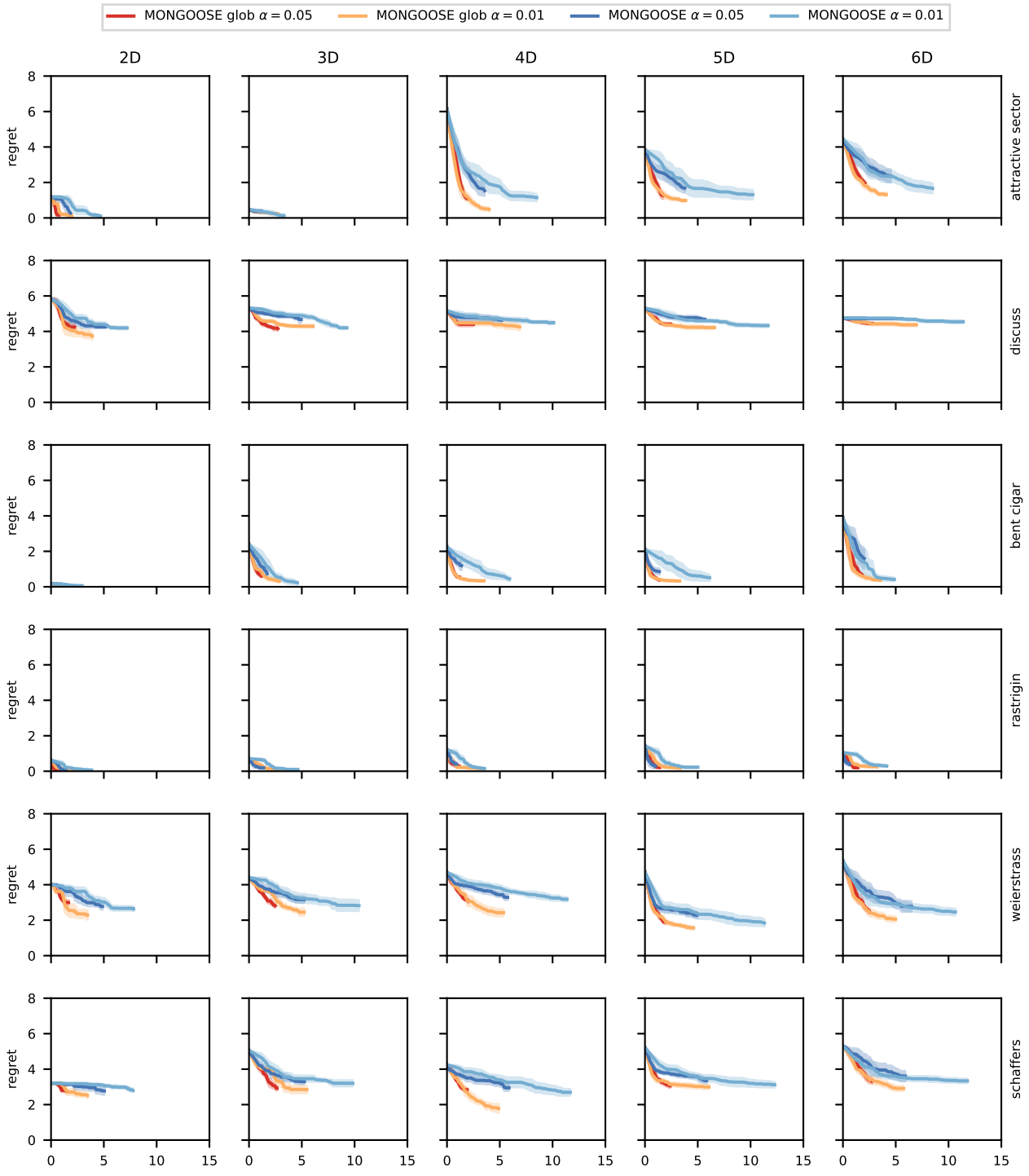


Figure 23: Individual COCO plots for Figure 11. COCO functions 13-18: sharp ridge function, different powers function, Rastrigin (non-separable) function, Weierstrrass function, Schaffers F7 function, Schaffers F7 (moderately ill-conditioned) function.

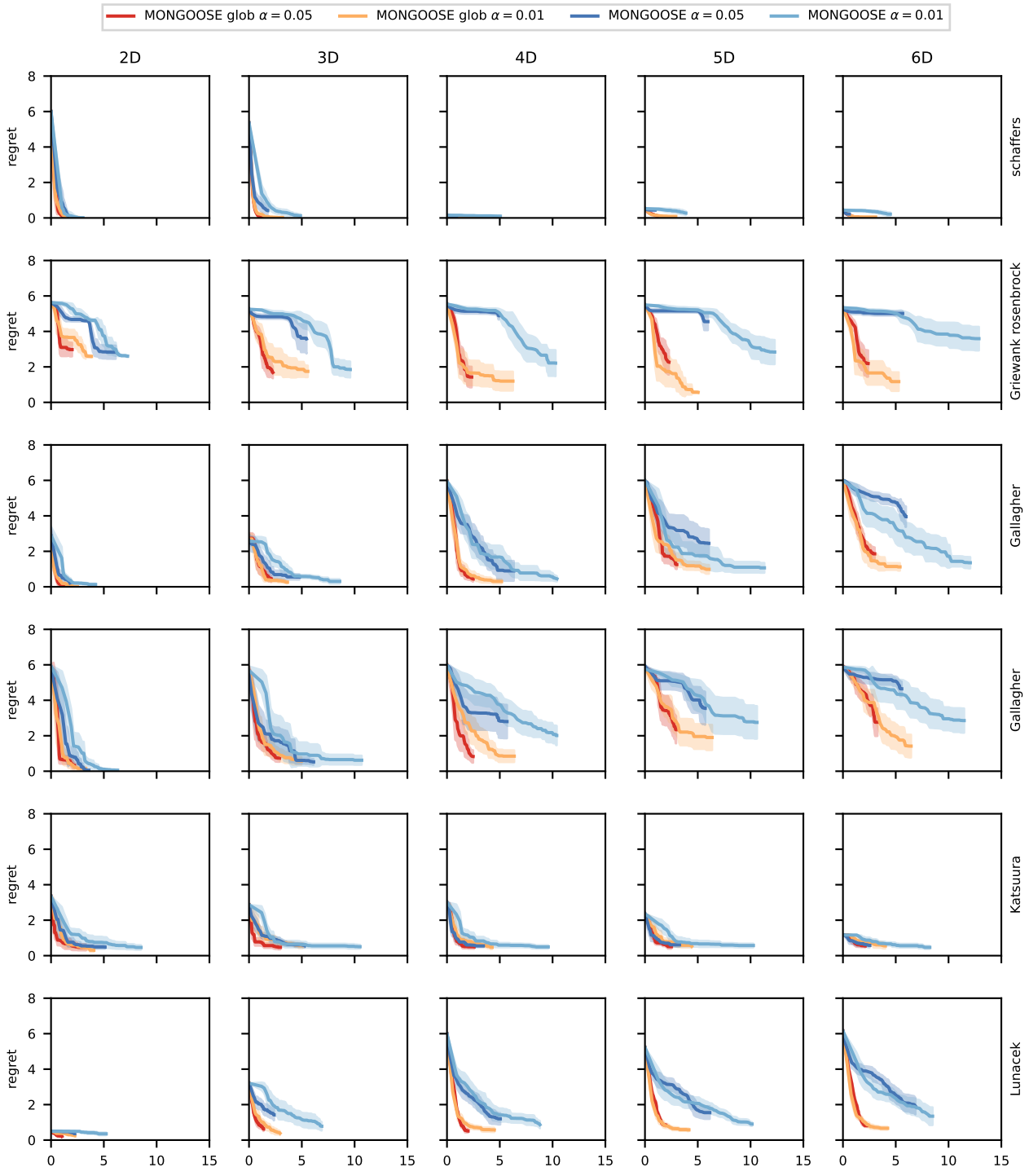


Figure 24: Individual COCO plots for Figure 11. COCO functions 19-24: composite Griewank-Rosenbrock function, Schwefel function, Gallagher's Gaussian 101-me peaks function, Gallagher's Gaussian 21-hi peaks function, Kastsuura function, Lunacek bi-Rastrigin function.

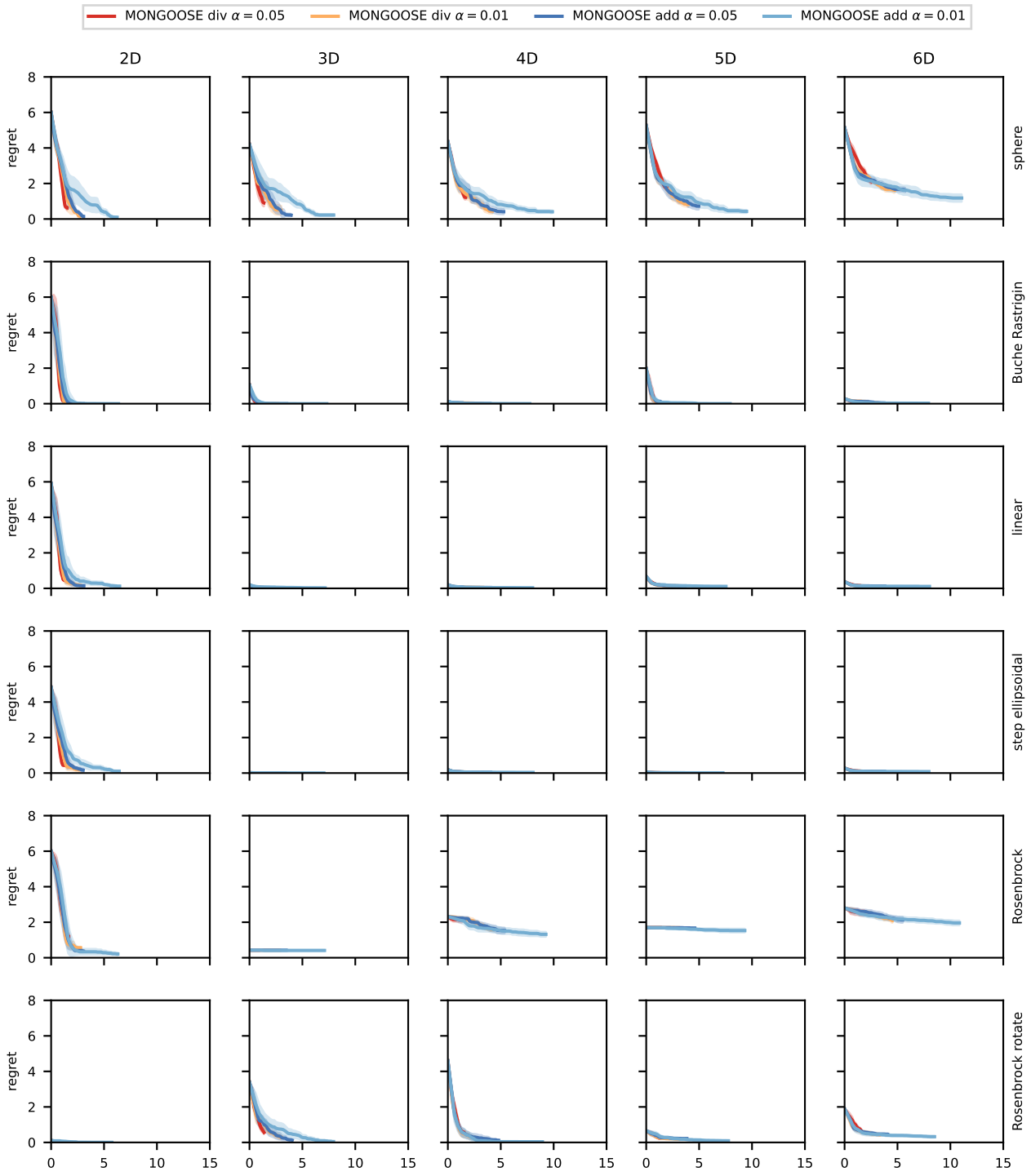


Figure 25: Individual COCO plots for Figure 7. COCO functions 1-6: sphere function, ellipsoidal function, Rastrigin function, Büche-Rastrigin function, linear slope, step ellipsoidal function.

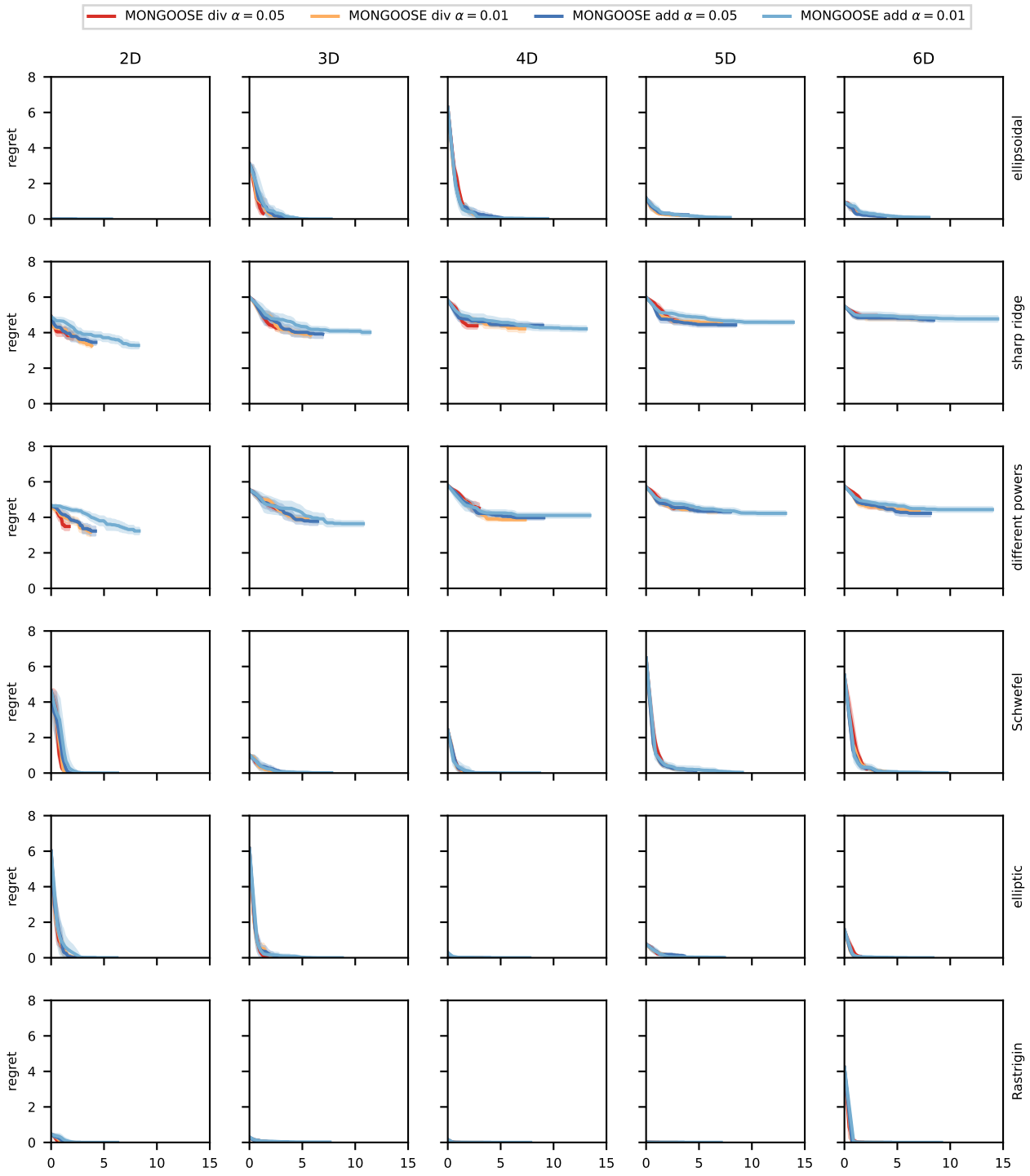


Figure 26: Individual COCO plots for Figure 7. COCO functions 7-12: attractive sector function, Rosenbrock (original) function, Rosenbrock (rotated) function, ellipsoidal (non-separable) function, discus function, bent cigar function.

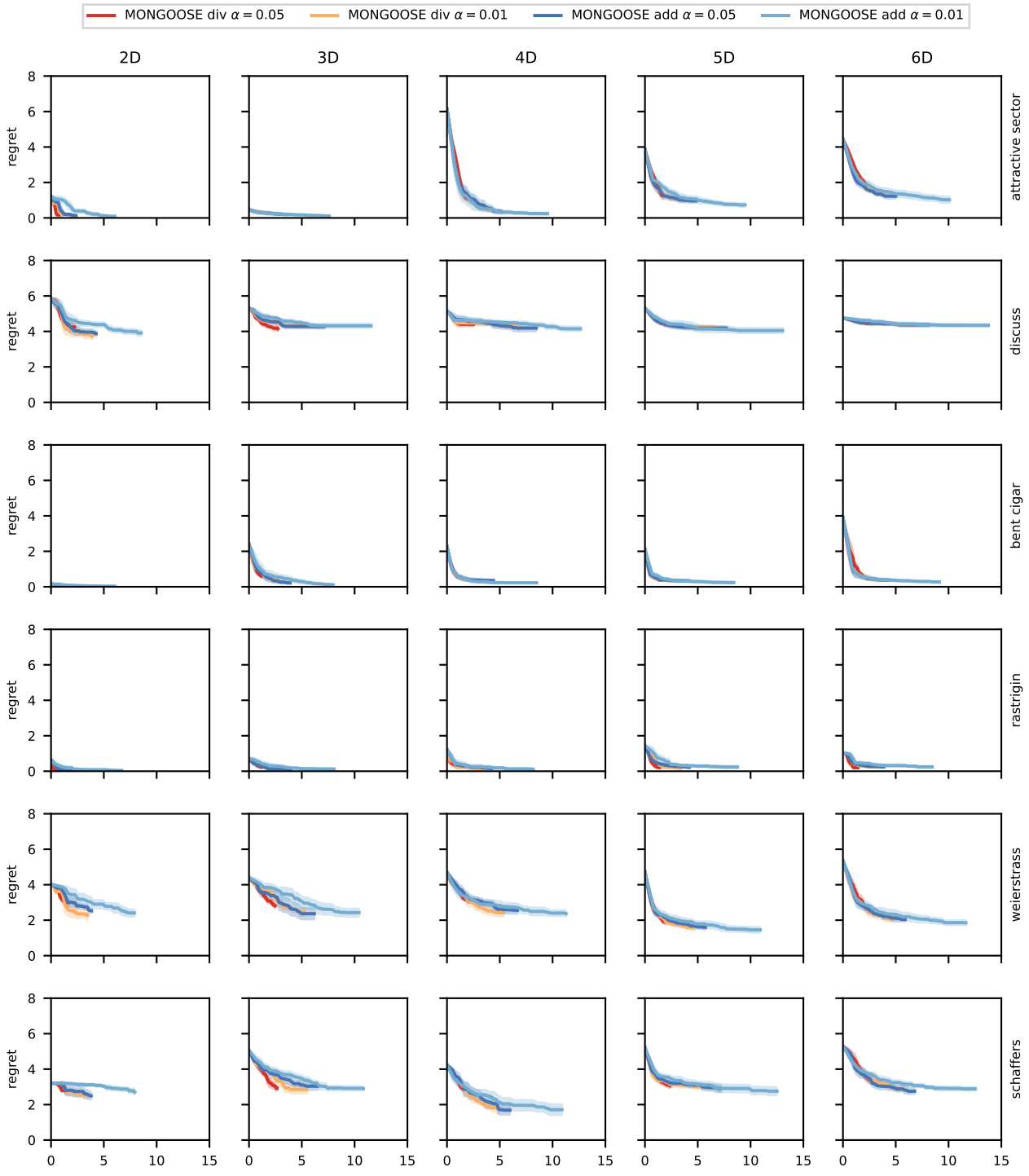


Figure 27: Individual COCO plots for Figure 7. COCO functions 13-18: sharp ridge function, different powers function, Rastrigin (non-separable) function, Weierstrrass function, Schaffers F7 function, Schaffers F7 (moderately ill-conditioned) function.

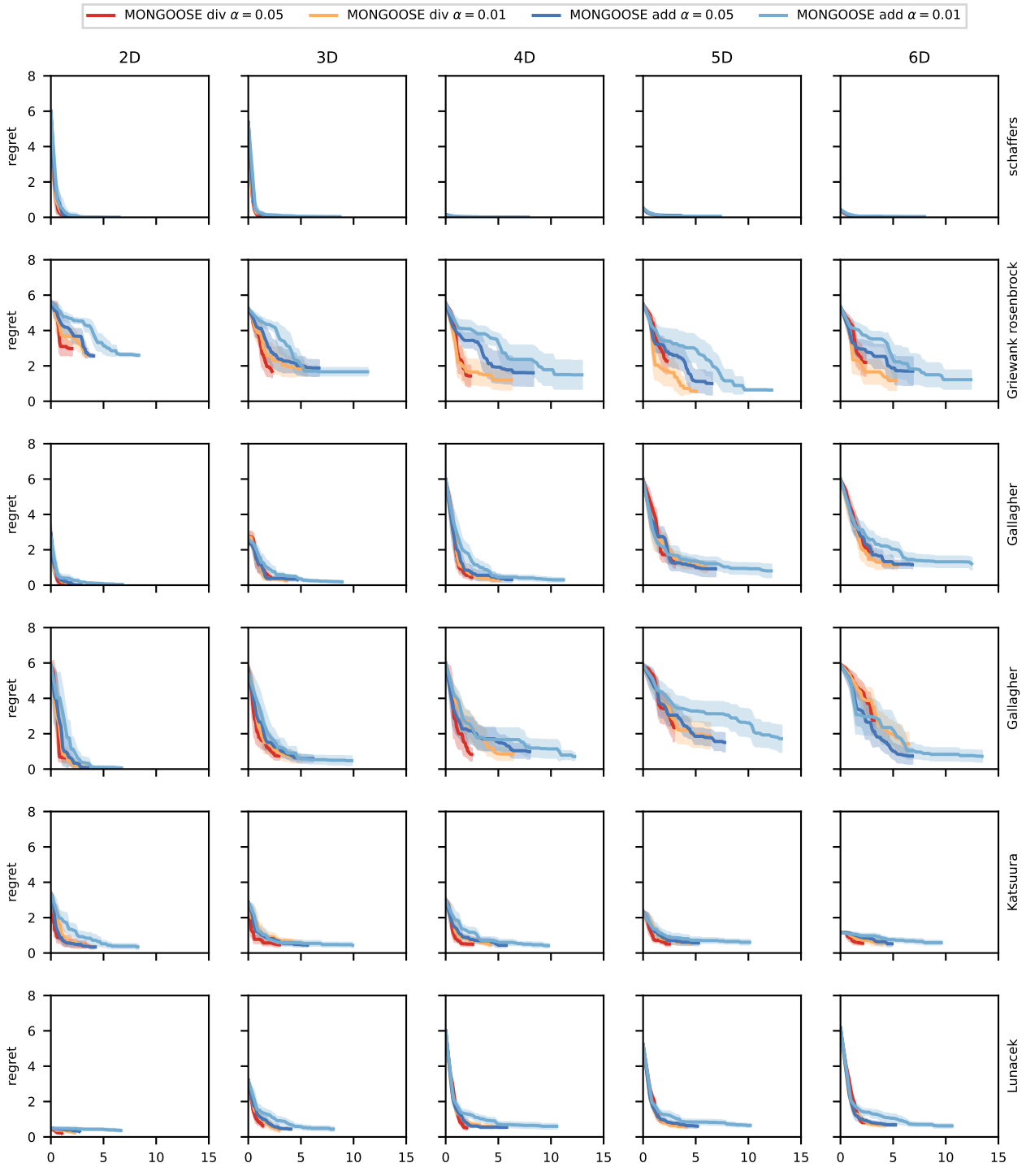


Figure 28: Individual COCO plots for Figure 7. COCO functions 19-24: composite Griewank-Rosenbrock function, Schwefel function, Gallagher's Gaussian 101-me peaks function, Gallagher's Gaussian 21-hi peaks function, Kastsuura function, Lunacek bi-Rastrigin function.

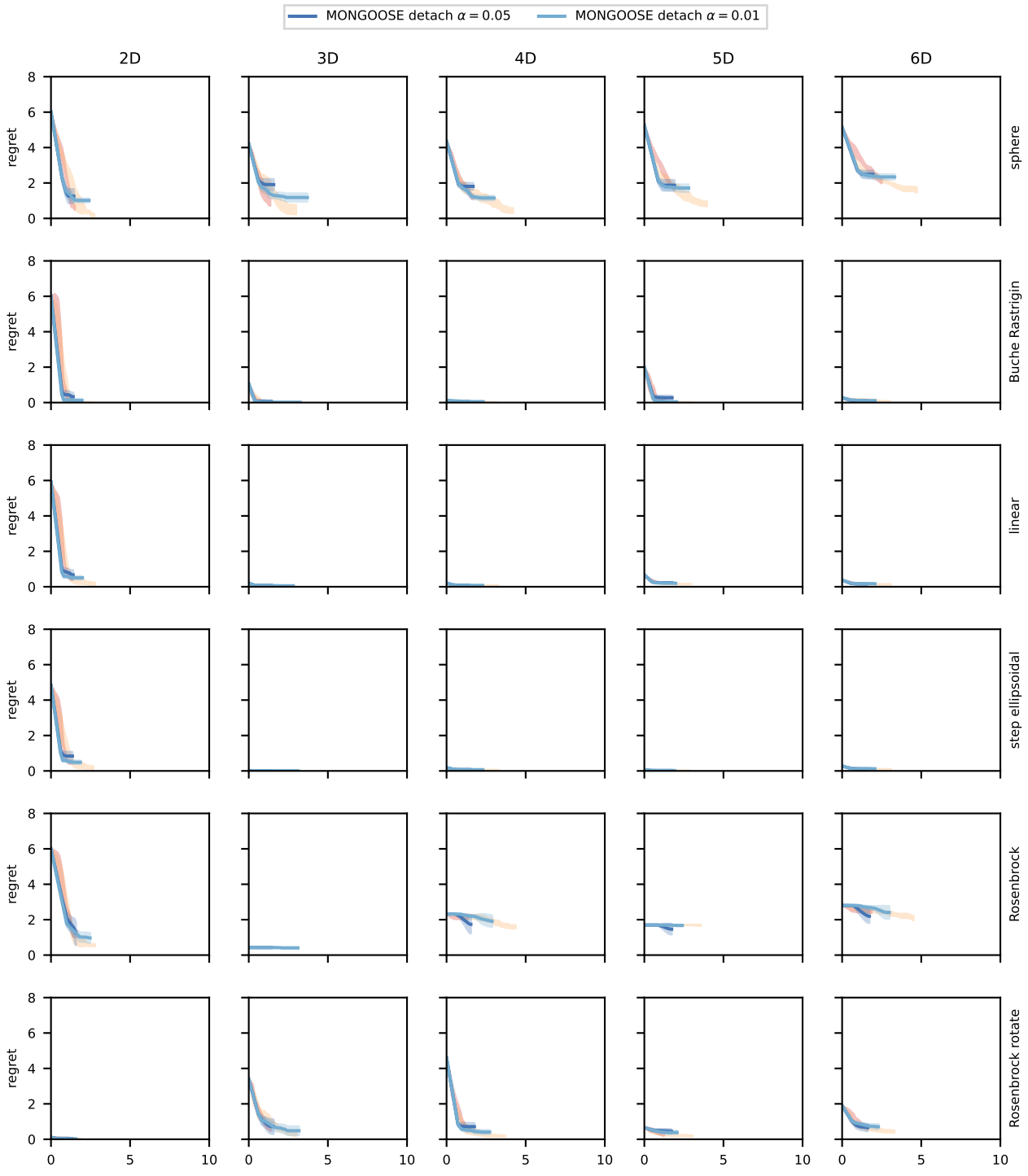


Figure 29: Individual COCO plots for Figure 7. COCO functions 1-6: sphere function, ellipsoidal function, Rastrigin function, Büche-Rastrigin function, linear slope, step ellipsoidal function.

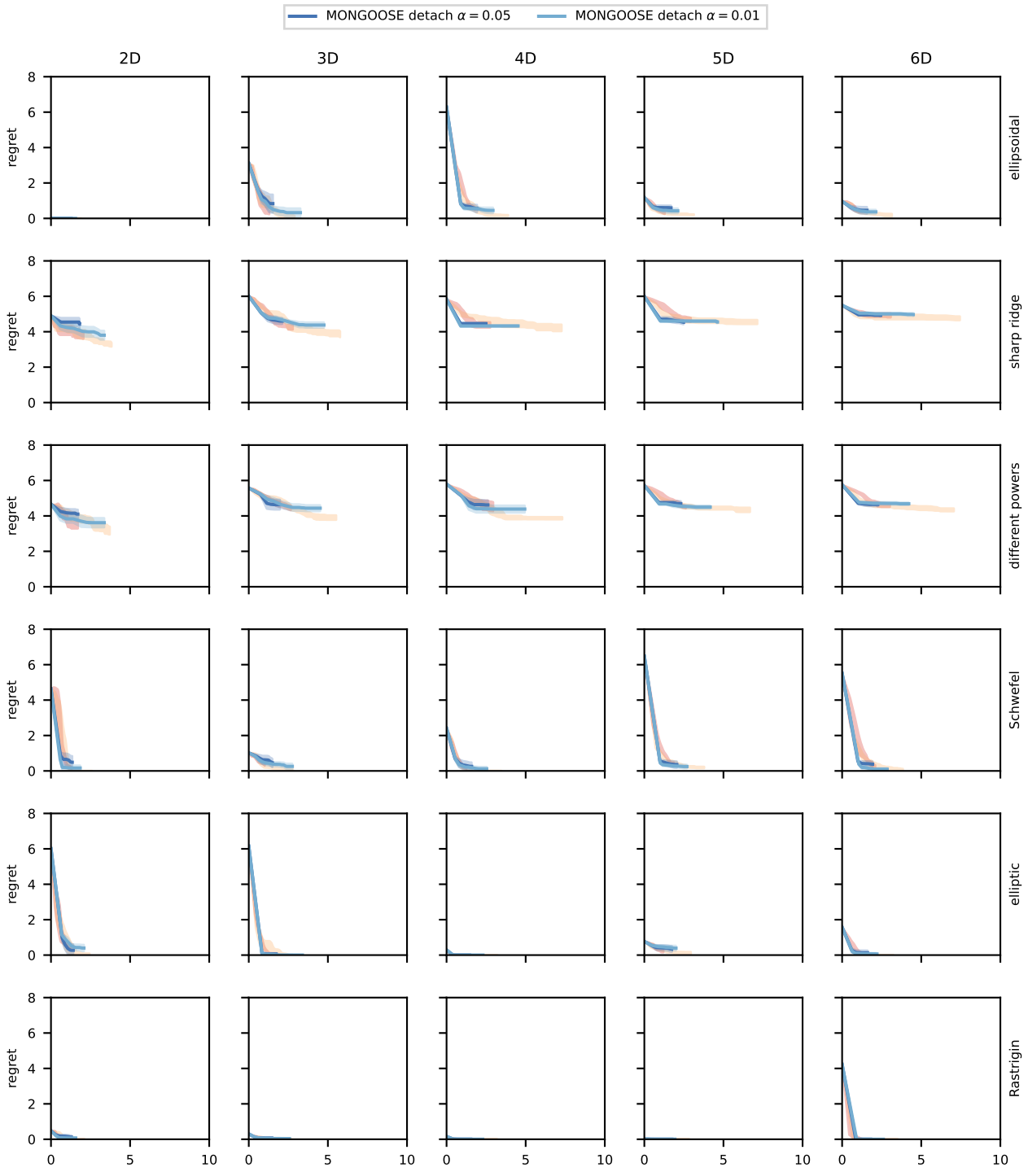


Figure 30: Individual COCO plots for Figure 7. COCO functions 7-12: attractive sector function, Rosenbrock (original) function, Rosenbrock (rotated) function, ellipsoidal (non-separable) function, discus function, bent cigar function.

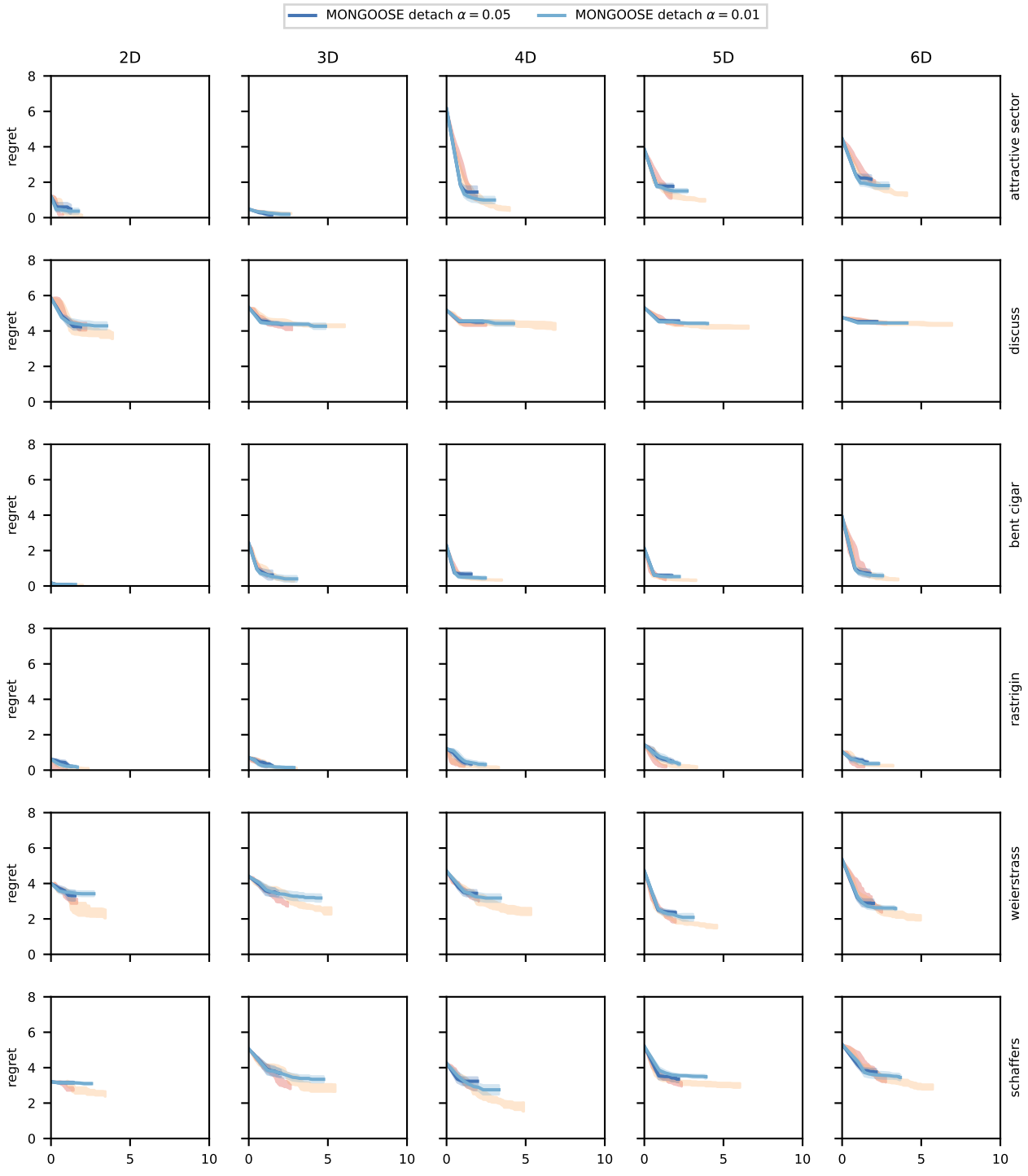


Figure 31: Individual COCO plots for Figure 7. COCO functions 13-18: sharp ridge function, different powers function, Rastrigin (non-separable) function, Weierstrrass function, Schaffers F7 function, Schaffers F7 (moderately ill-conditioned) function.

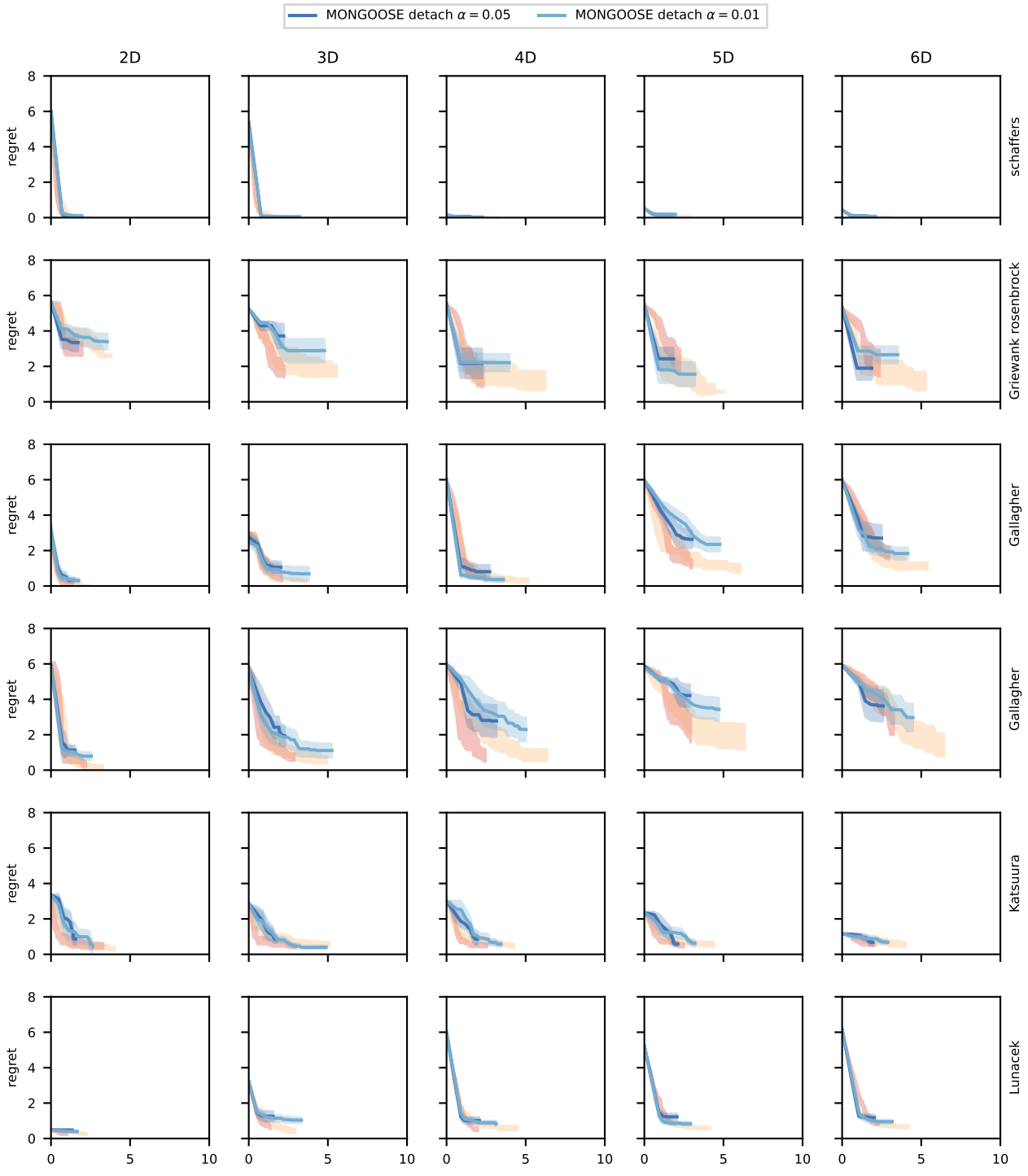


Figure 32: Individual COCO plots for Figure 8. COCO functions 19-24: composite Griewank-Rosenbrock function, Schwefel function, Gallagher's Gaussian 101-me peaks function, Gallagher's Gaussian 21-hi peaks function, Kastsuura function, Lunacek bi-Rastrigin function.

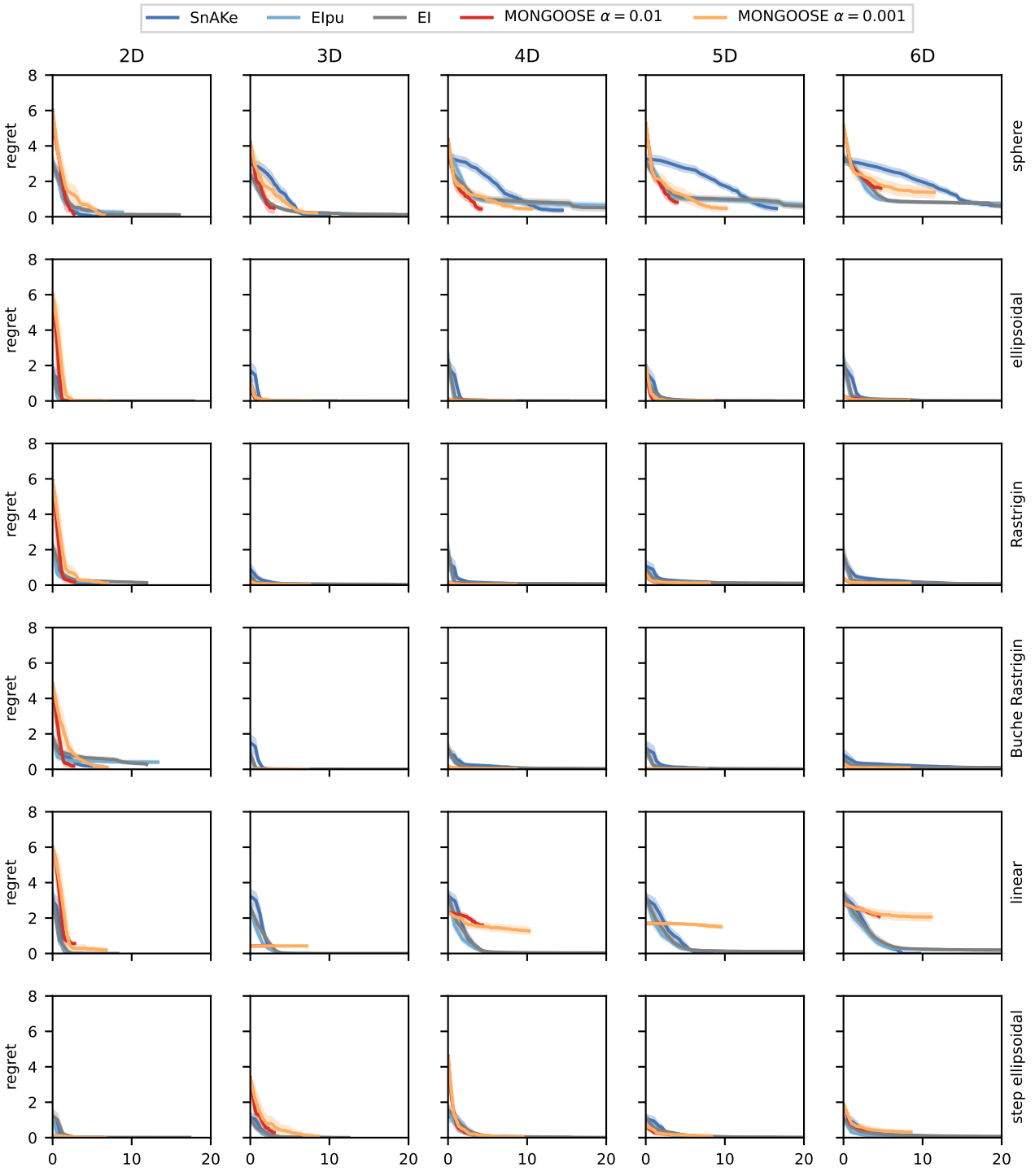


Figure 33: Individual COCO plots for Figure 14. COCO functions 1-6: sphere function, ellipsoidal function, Rastrigin function, Büche-Rastrigin function, linear slope, step ellipsoidal function.

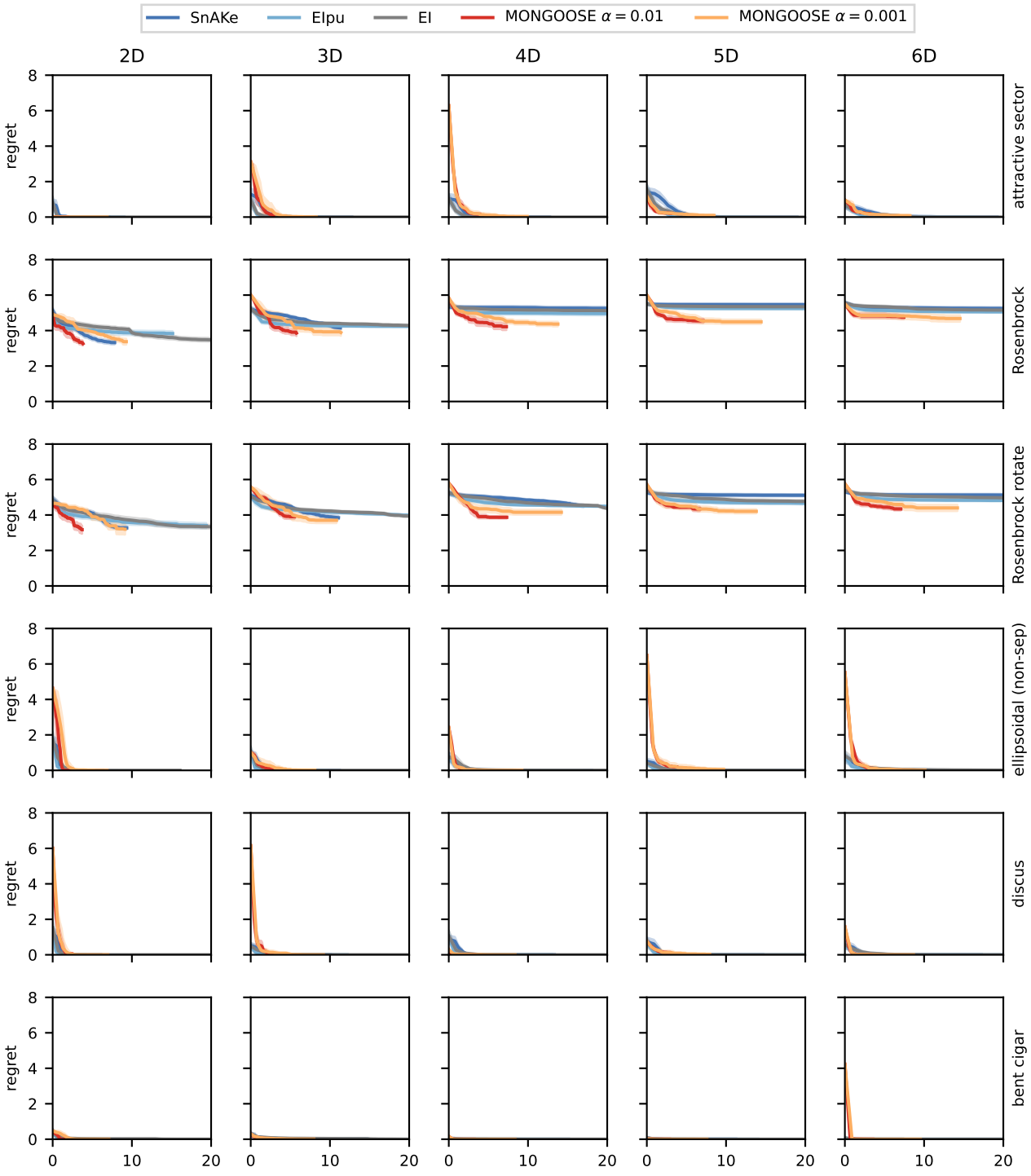


Figure 34: Individual COCO plots for Figure 14. COCO functions 7-12: attractive sector function, Rosenbrock (original) function, Rosenbrock (rotated) function, ellipsoidal (non-separable) function, discus function, bent cigar function.

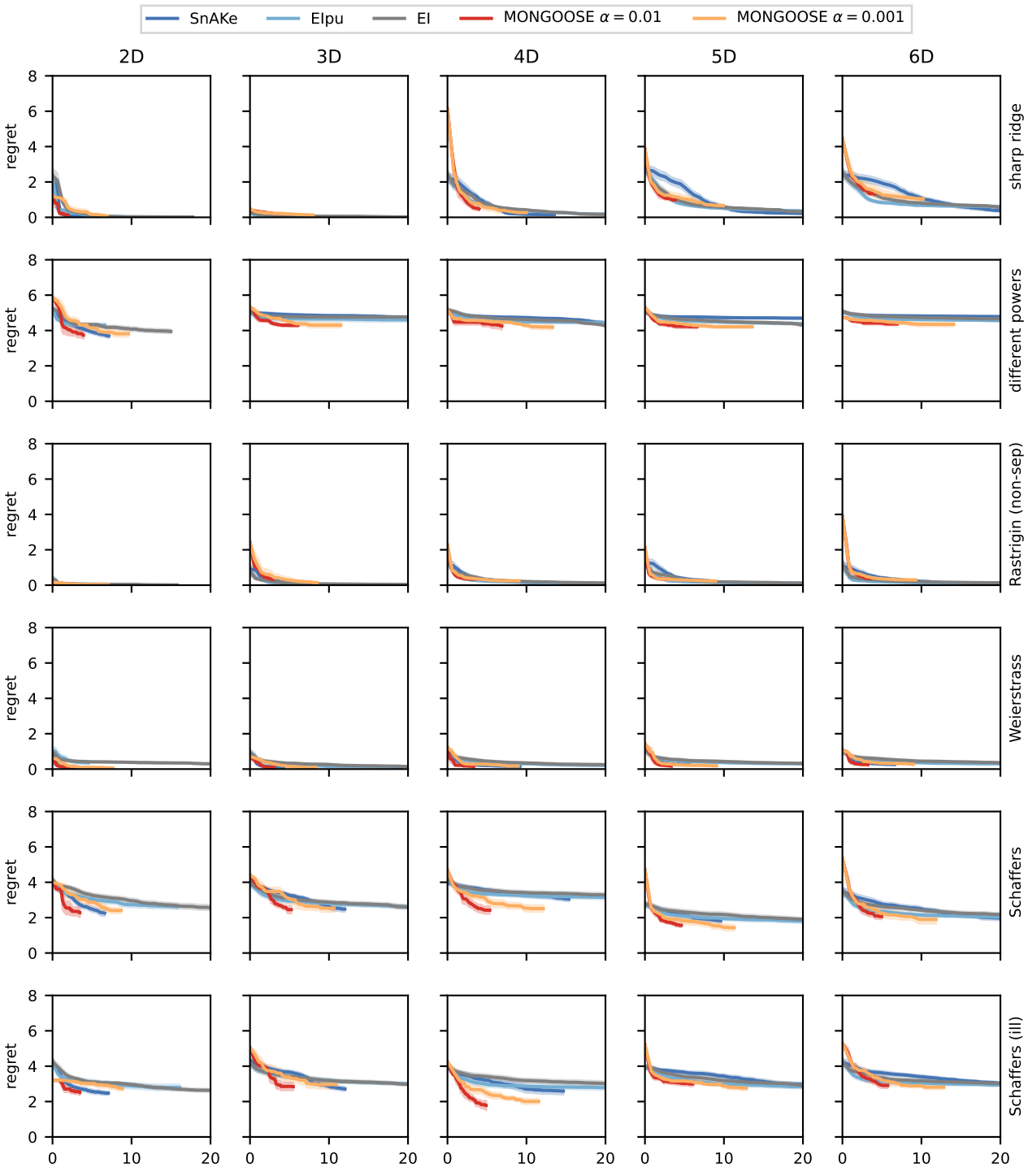


Figure 35: Individual COCO plots for Figure 14. COCO functions 13-18: sharp ridge function, different powers function, Rastrigin (non-separable) function, Weierstrrass function, Schaffers F7 function, Schaffers F7 (moderately ill-conditioned) function.

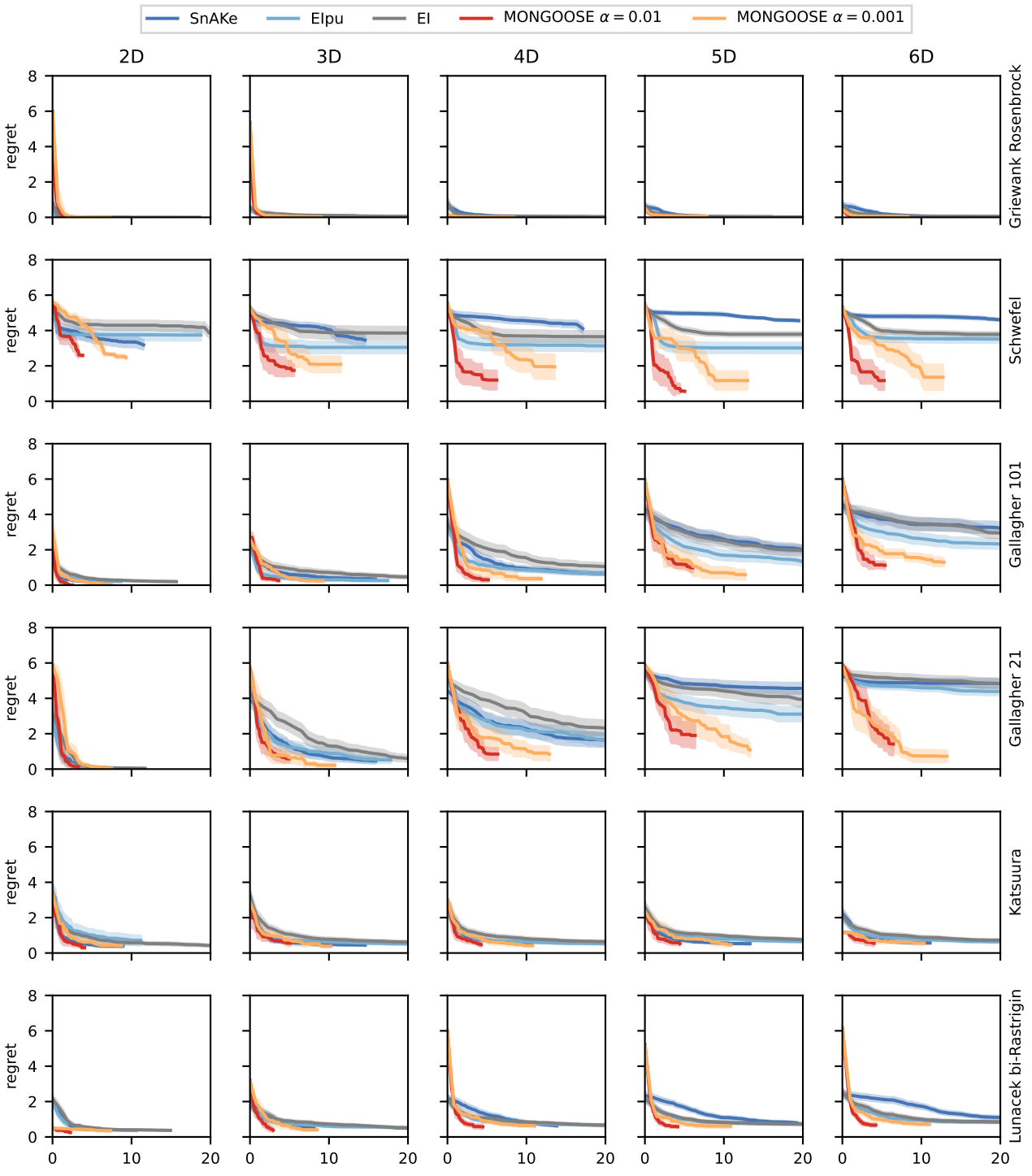


Figure 36: Individual COCO plots for Figure 14. COCO functions 19-24: composite Griewank-Rosenbrock function, Schwefel function, Gallagher's Gaussian 101-me peaks function, Gallagher's Gaussian 21-hi peaks function, Kastsuura function, Lunacek bi-Rastrigin function.

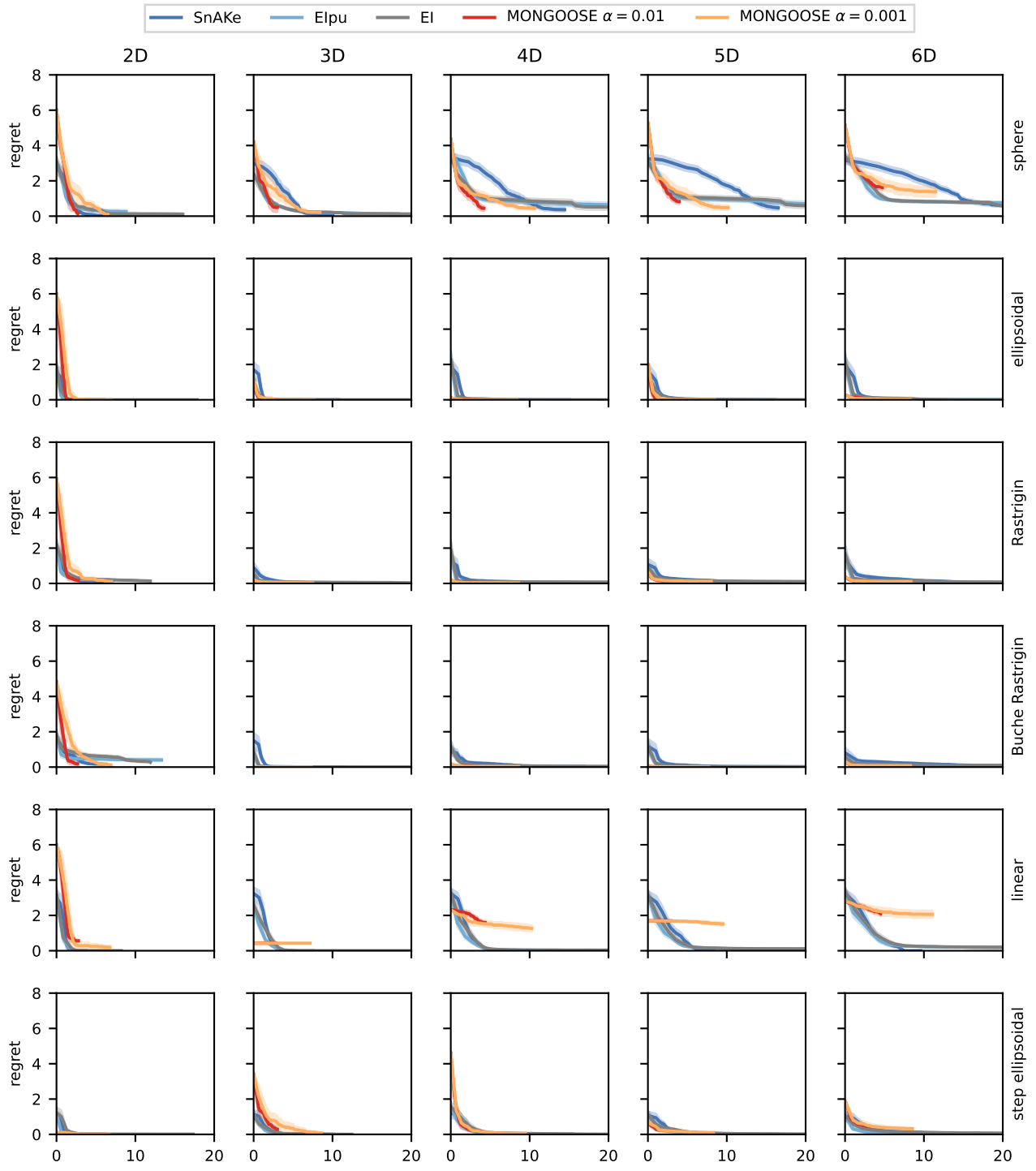


Figure 37: Individual COCO plots for Figure 37. COCO functions 1-6: sphere function, ellipsoidal function, Rastrigin function, Büche-Rastrigin function, linear slope, step ellipsoidal function.

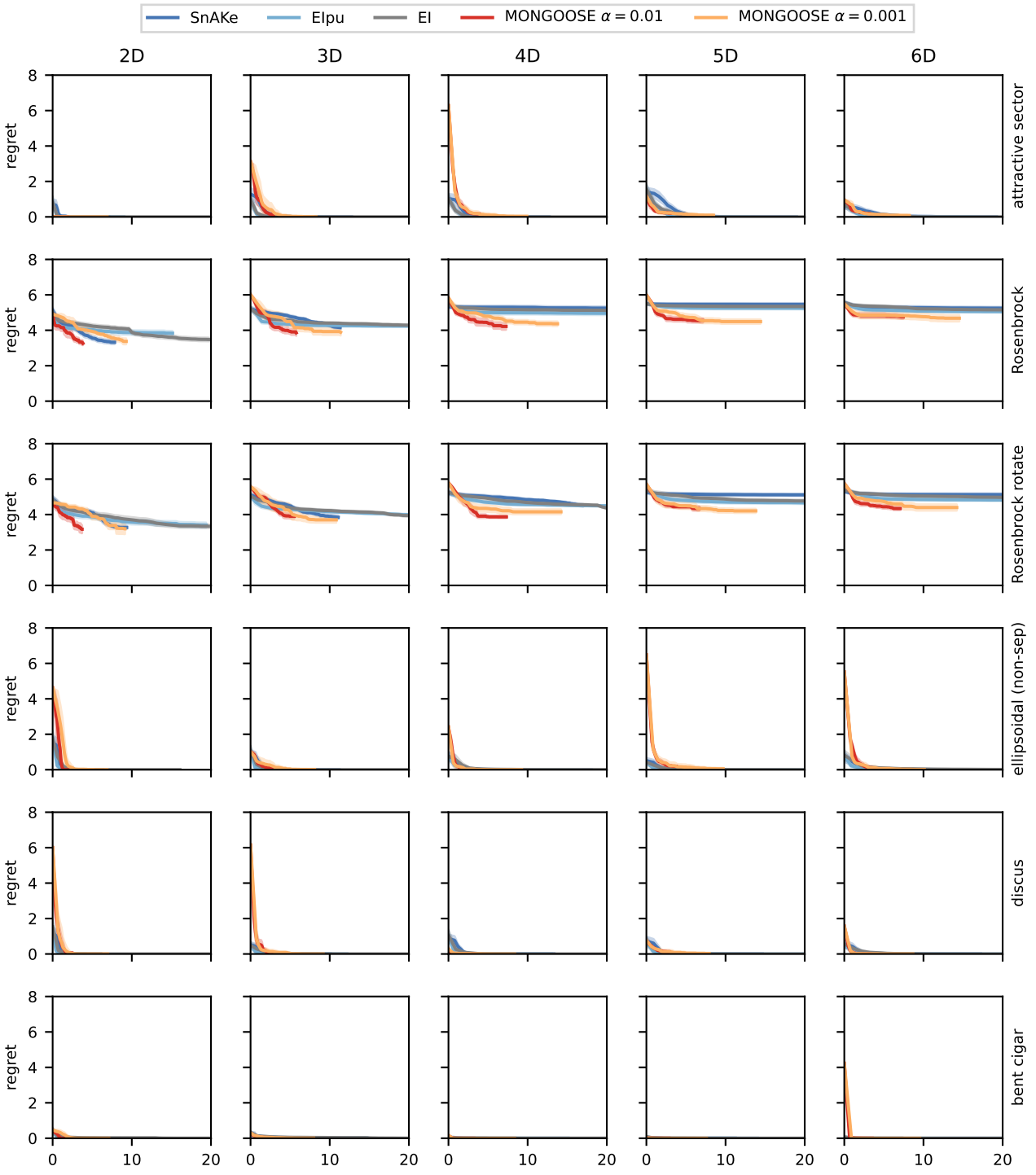


Figure 38: Individual COCO plots for Figure 37. COCO functions 7-12: attractive sector function, Rosenbrock (original) function, Rosenbrock (rotated) function, ellipsoidal (non-separable) function, discus function, bent cigar function.

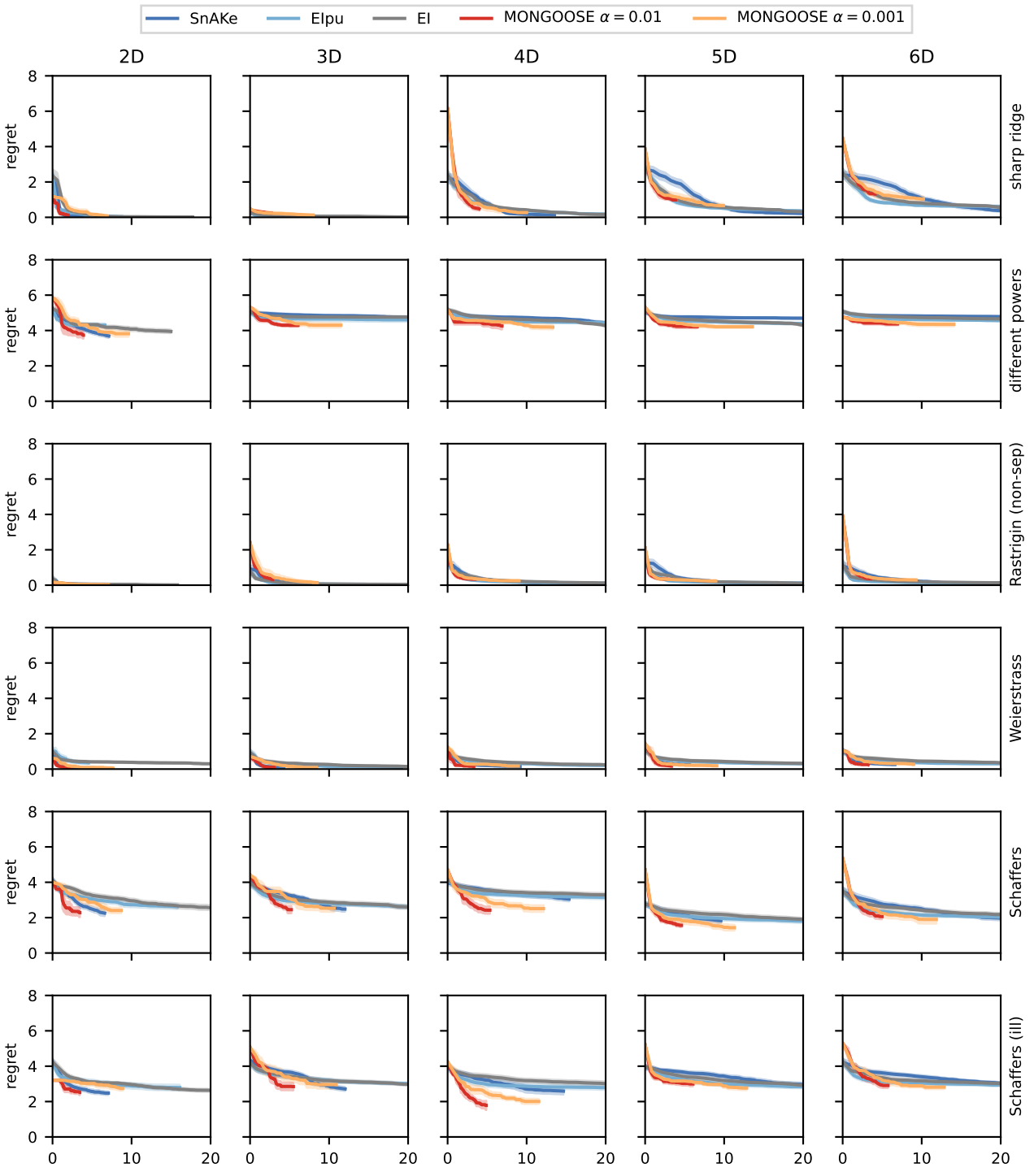


Figure 39: Individual COCO plots for Figure 37. COCO functions 13-18: sharp ridge function, different powers function, Rastrigin (non-separable) function, Weierstrrass function, Schaffers F7 function, Schaffers F7 (moderately ill-conditioned) function.

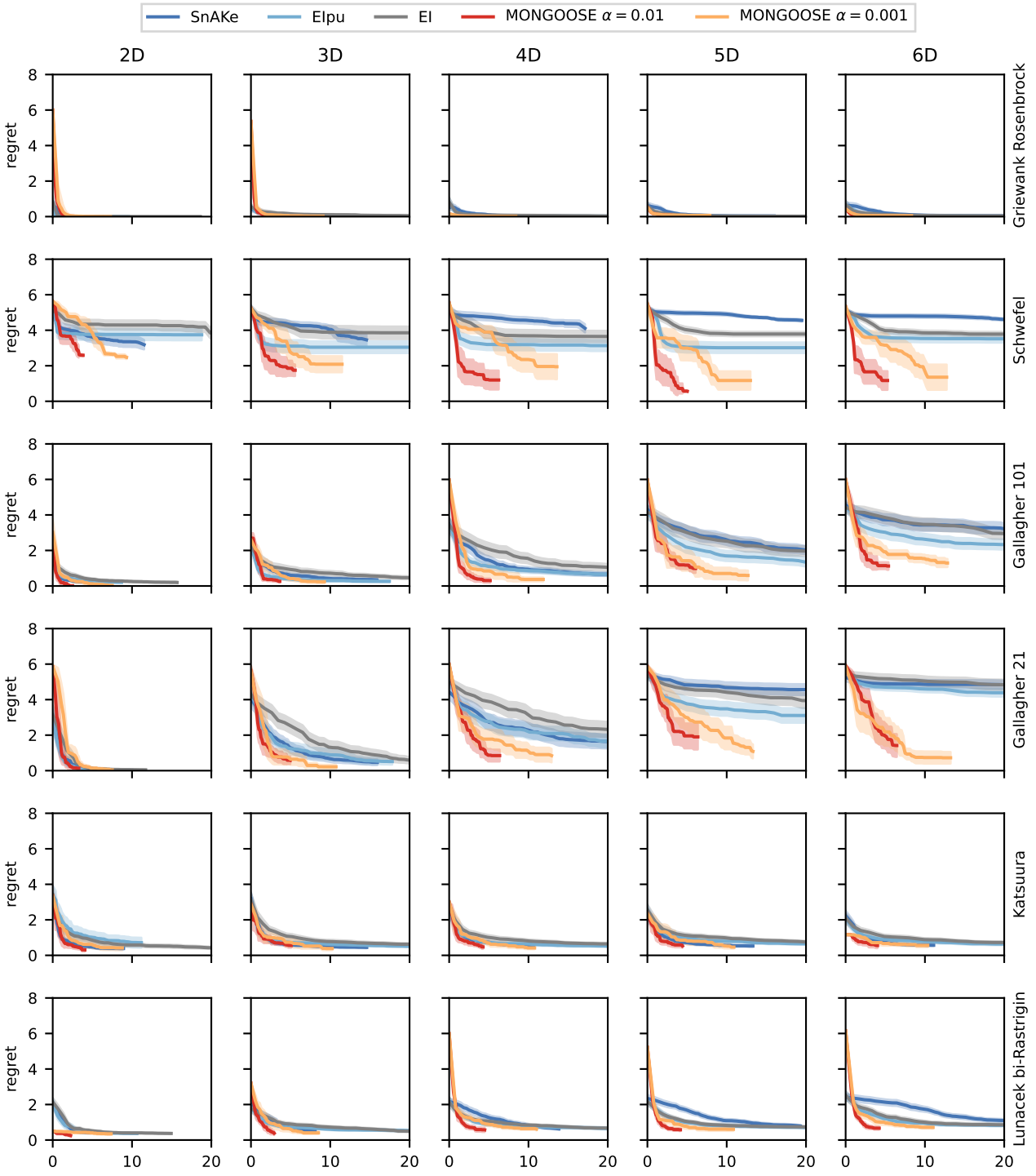


Figure 40: Individual COCO plots for Figure 37. COCO functions 19-24: composite Griewank-Rosenbrock function, Schwefel function, Gallagher's Gaussian 101-me peaks function, Gallagher's Gaussian 21-hi peaks function, Kastsuura function, Lunacek bi-Rastrigin function.

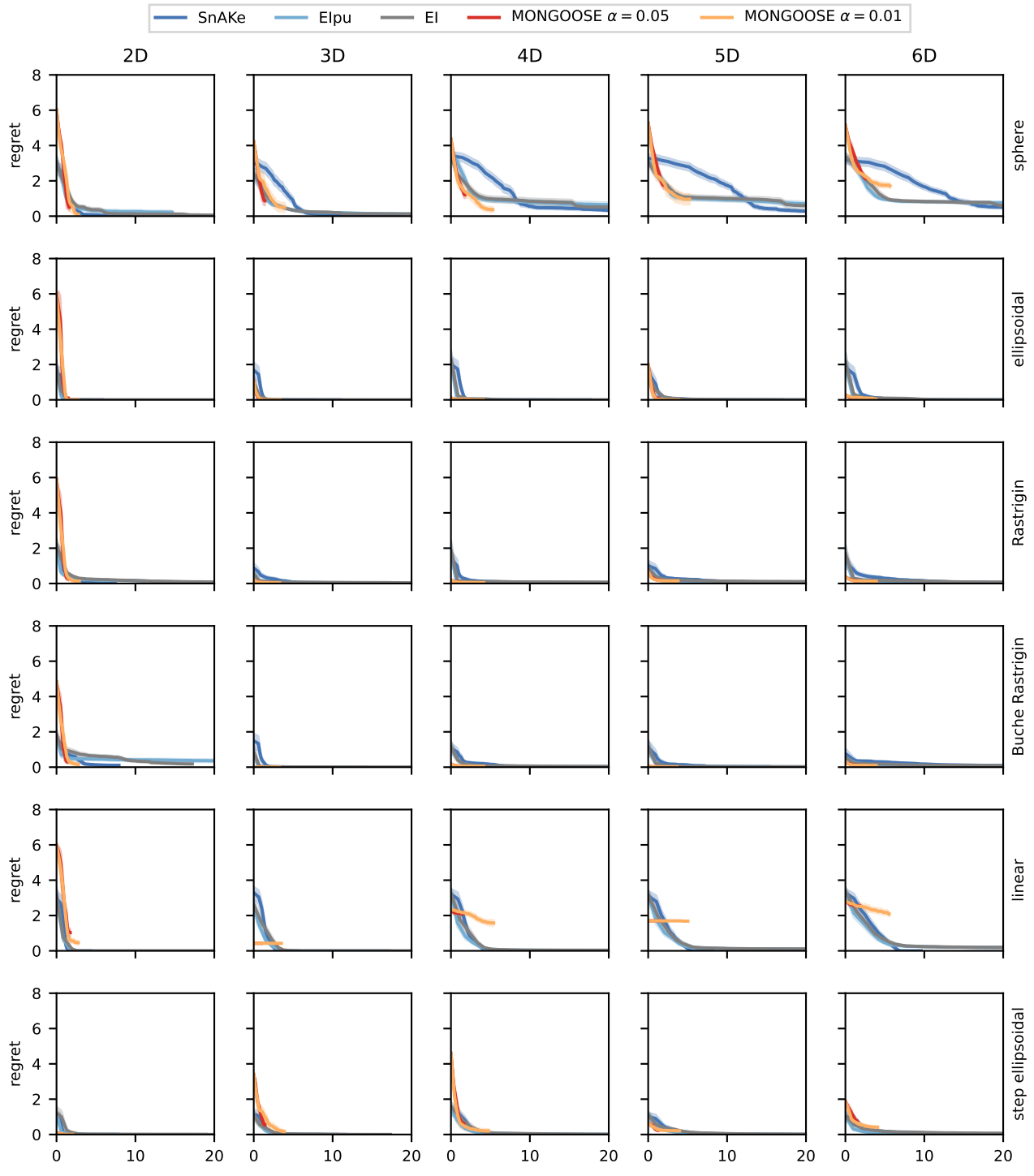


Figure 41: Individual COCO plots for Figure 18. COCO functions 1-6: sphere function, ellipsoidal function, Rastrigin function, Büche-Rastrigin function, linear slope, step ellipsoidal function.

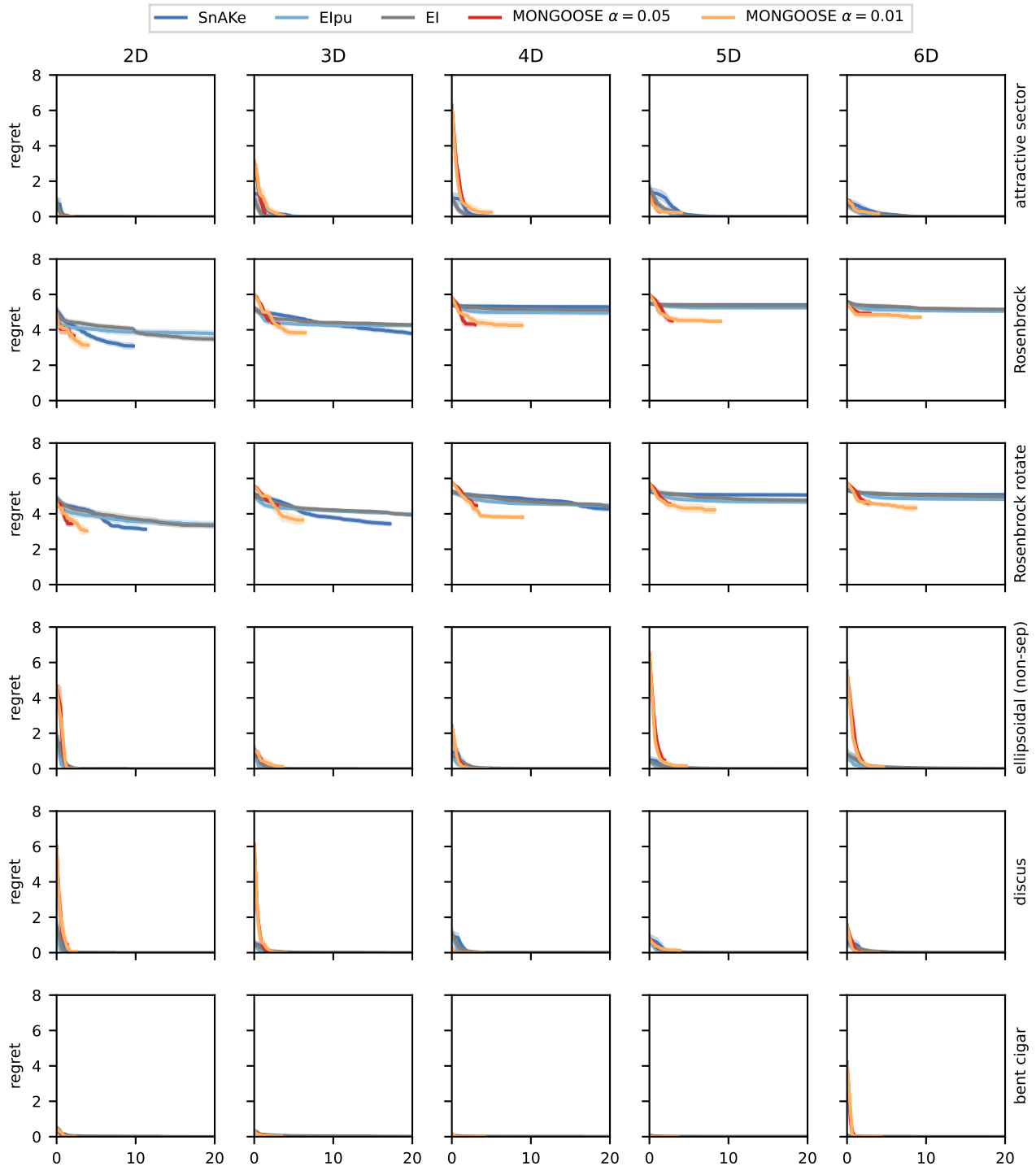


Figure 42: Individual COCO plots for Figure 18. COCO functions 7-12: attractive sector function, Rosenbrock (original) function, Rosenbrock (rotated) function, ellipsoidal (non-separable) function, discus function, bent cigar function.

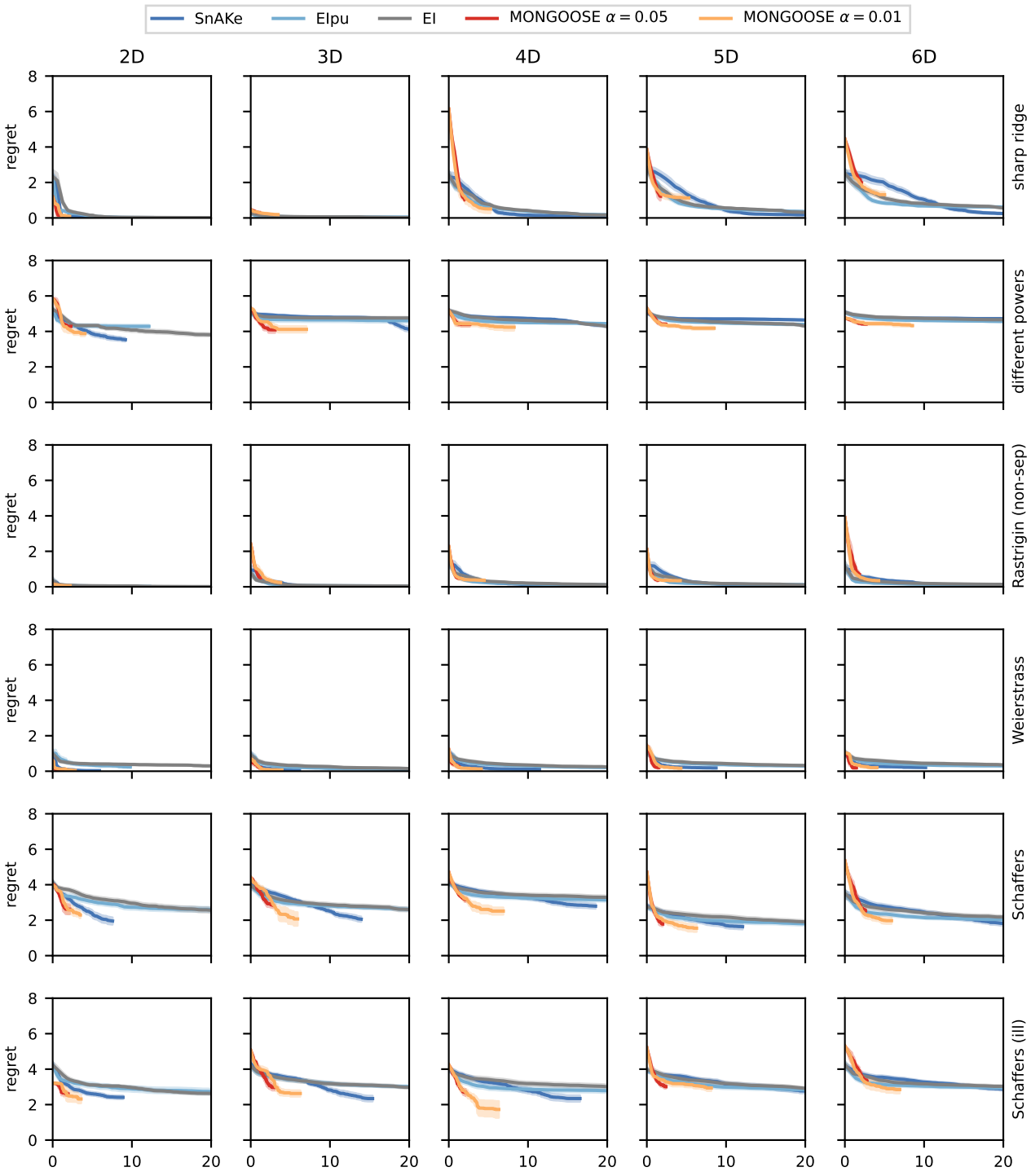


Figure 43: Individual COCO plots for Figure 18. COCO functions 13-18: sharp ridge function, different powers function, Rastrigin (non-separable) function, Weierstrrass function, Schaffers F7 function, Schaffers F7 (moderately ill-conditioned) function.

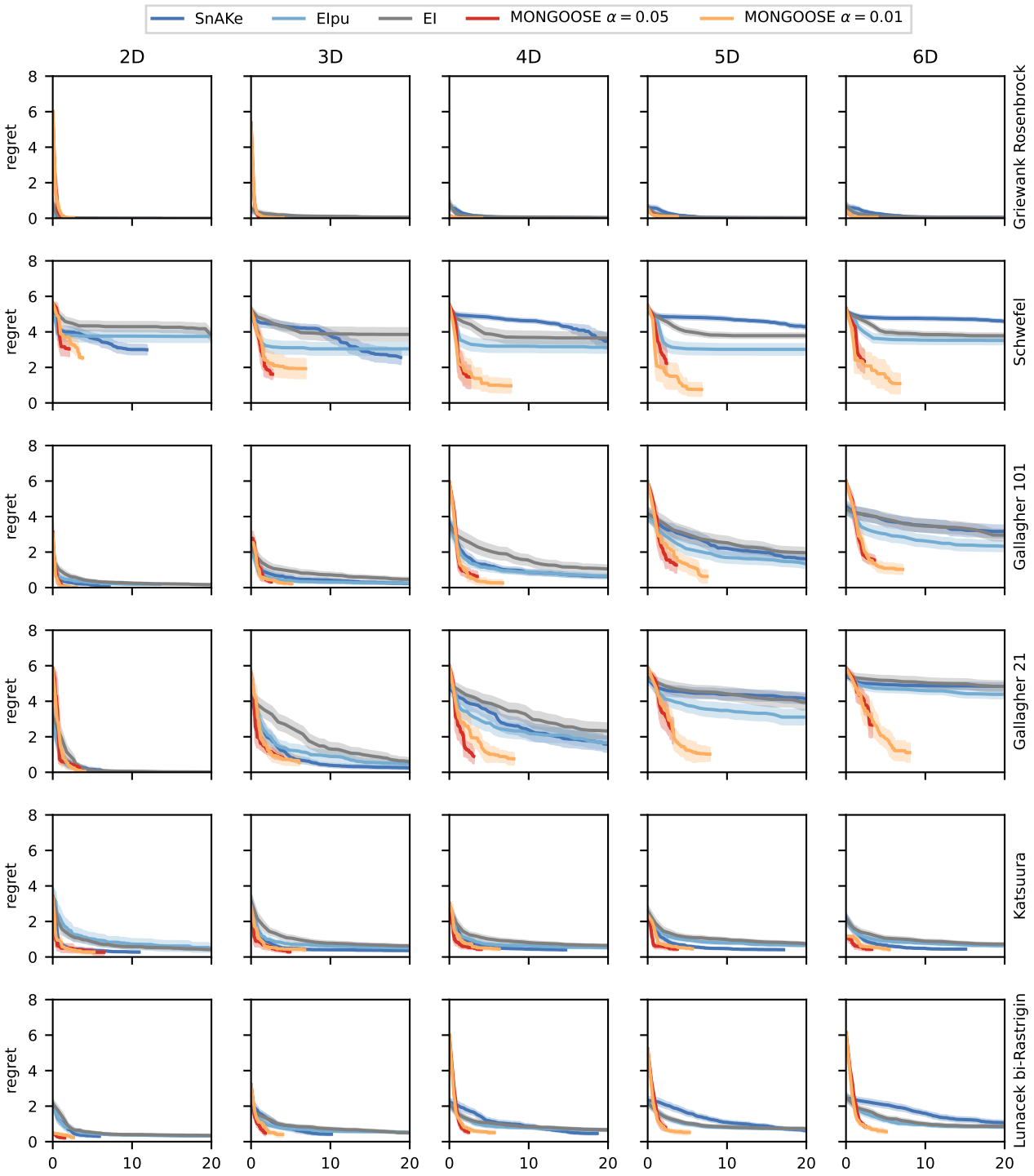


Figure 44: Individual COCO plots for Figure 18. COCO functions 19-24: composite Griewank-Rosenbrock function, Schwefel function, Gallagher's Gaussian 101-me peaks function, Gallagher's Gaussian 21-hi peaks function, Kastsuura function, Lunacek bi-Rastrigin function.

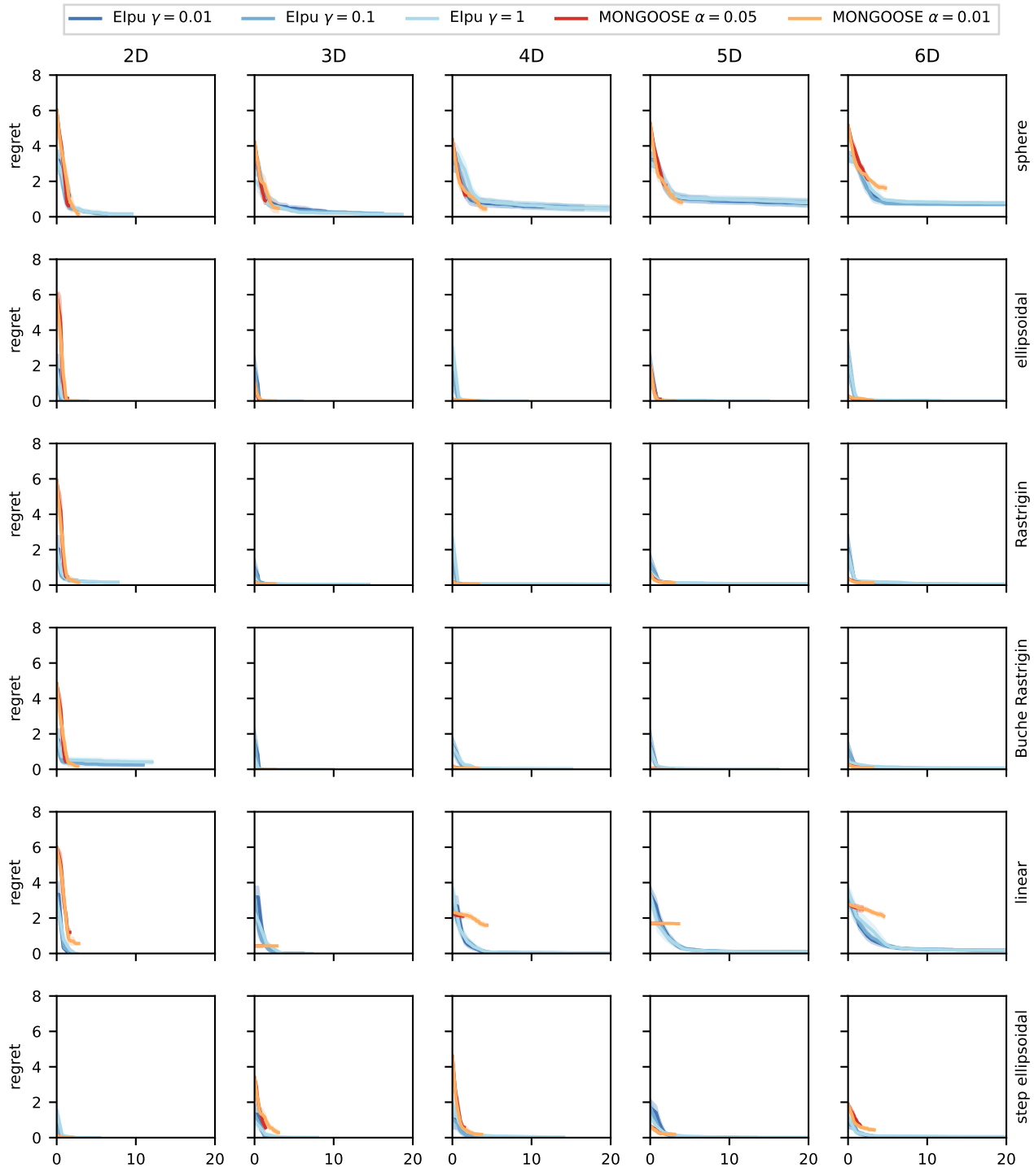


Figure 45: Individual COCO plots for Figure 19. COCO functions 1-6: sphere function, ellipsoidal function, Rastrigin function, Büche-Rastrigin function, linear slope, step ellipsoidal function.

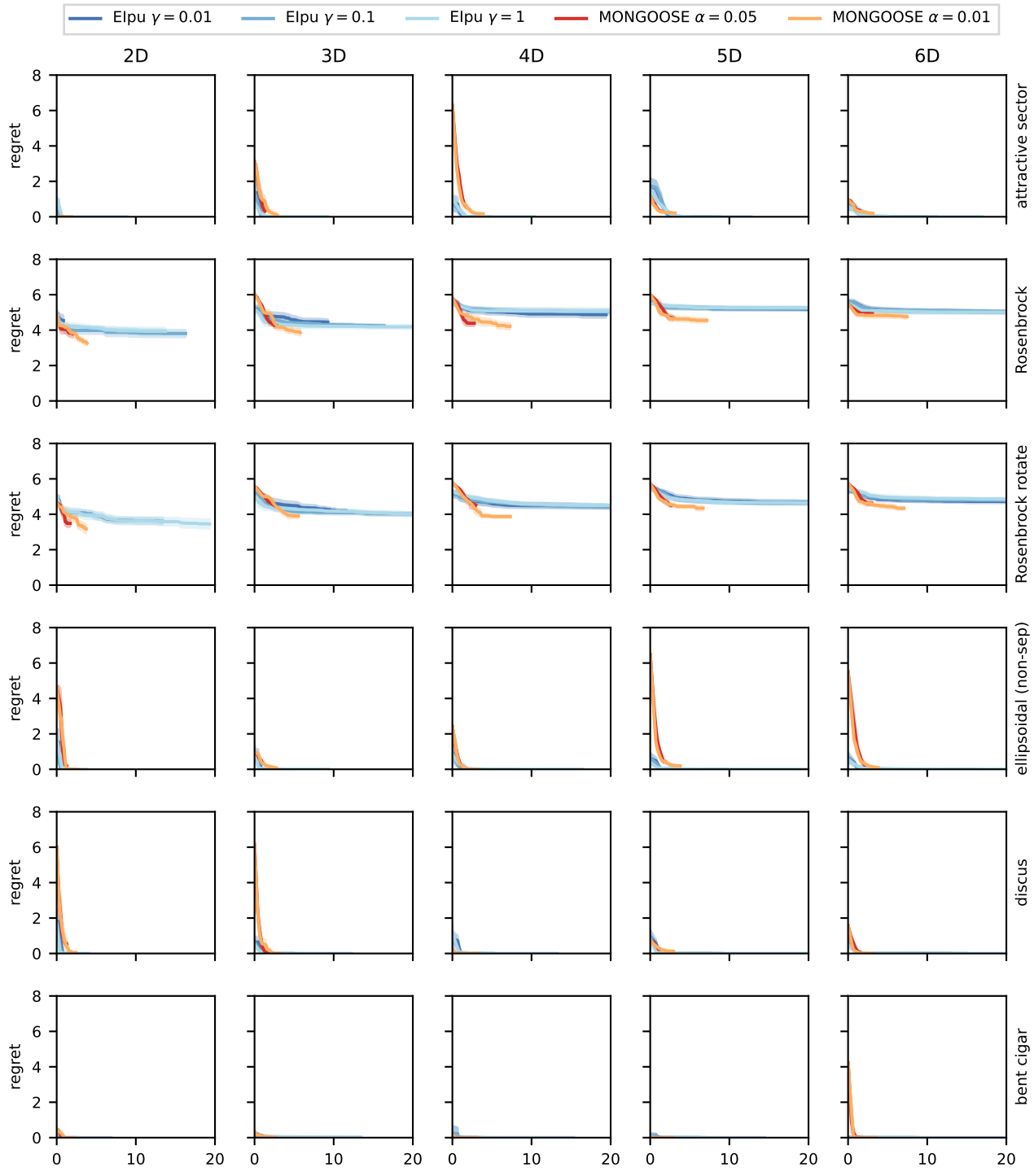


Figure 46: Individual COCO plots for Figure 19. COCO functions 7-12: attractive sector function, Rosenbrock (original) function, Rosenbrock (rotated) function, ellipsoidal (non-separable) function, discus function, bent cigar function.

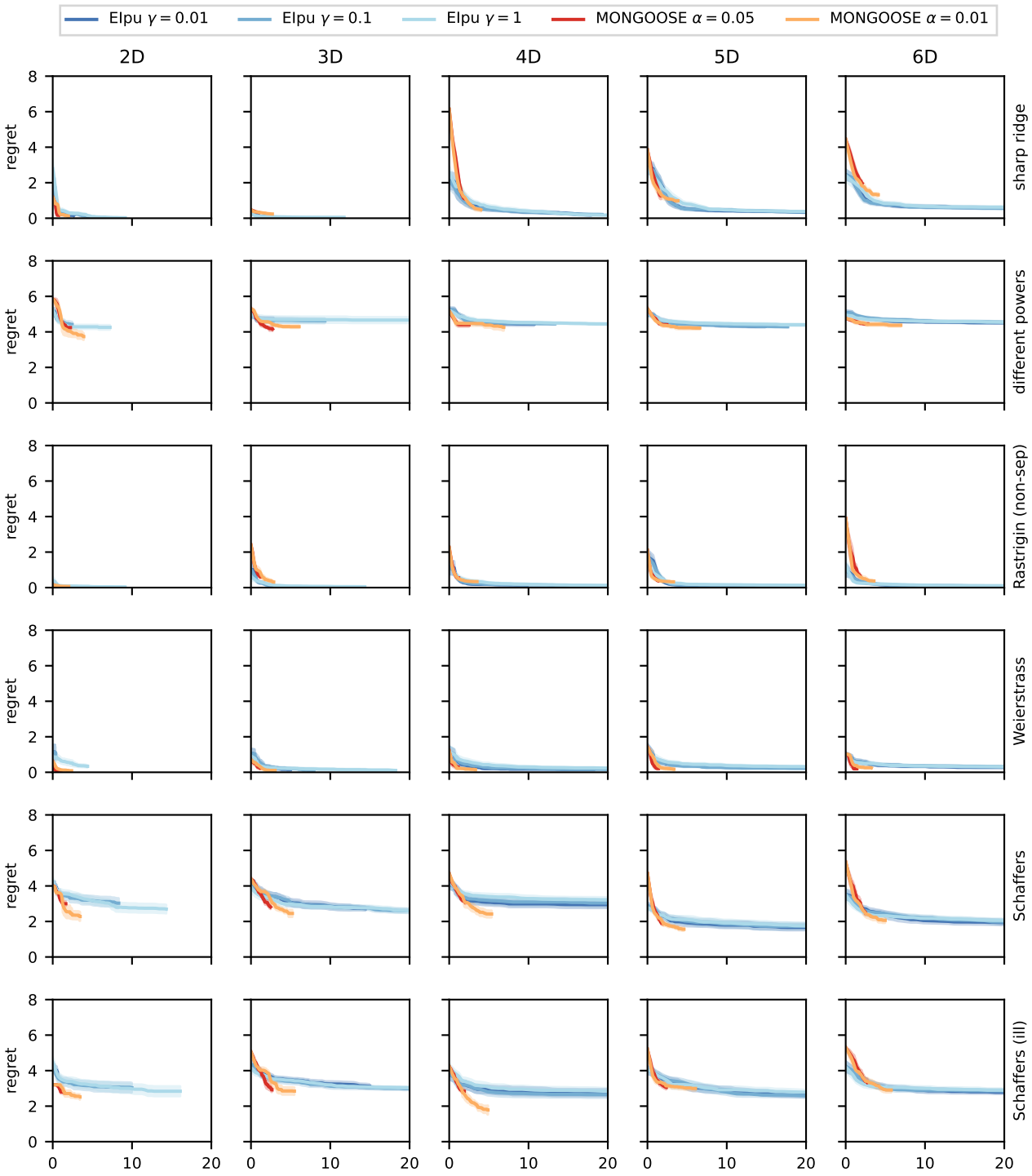


Figure 47: Individual COCO plots for Figure 19. COCO functions 13-18: sharp ridge function, different powers function, Rastrigin (non-separable) function, Weierstrrass function, Schaffers F7 function, Schaffers F7 (moderately ill-conditioned) function.

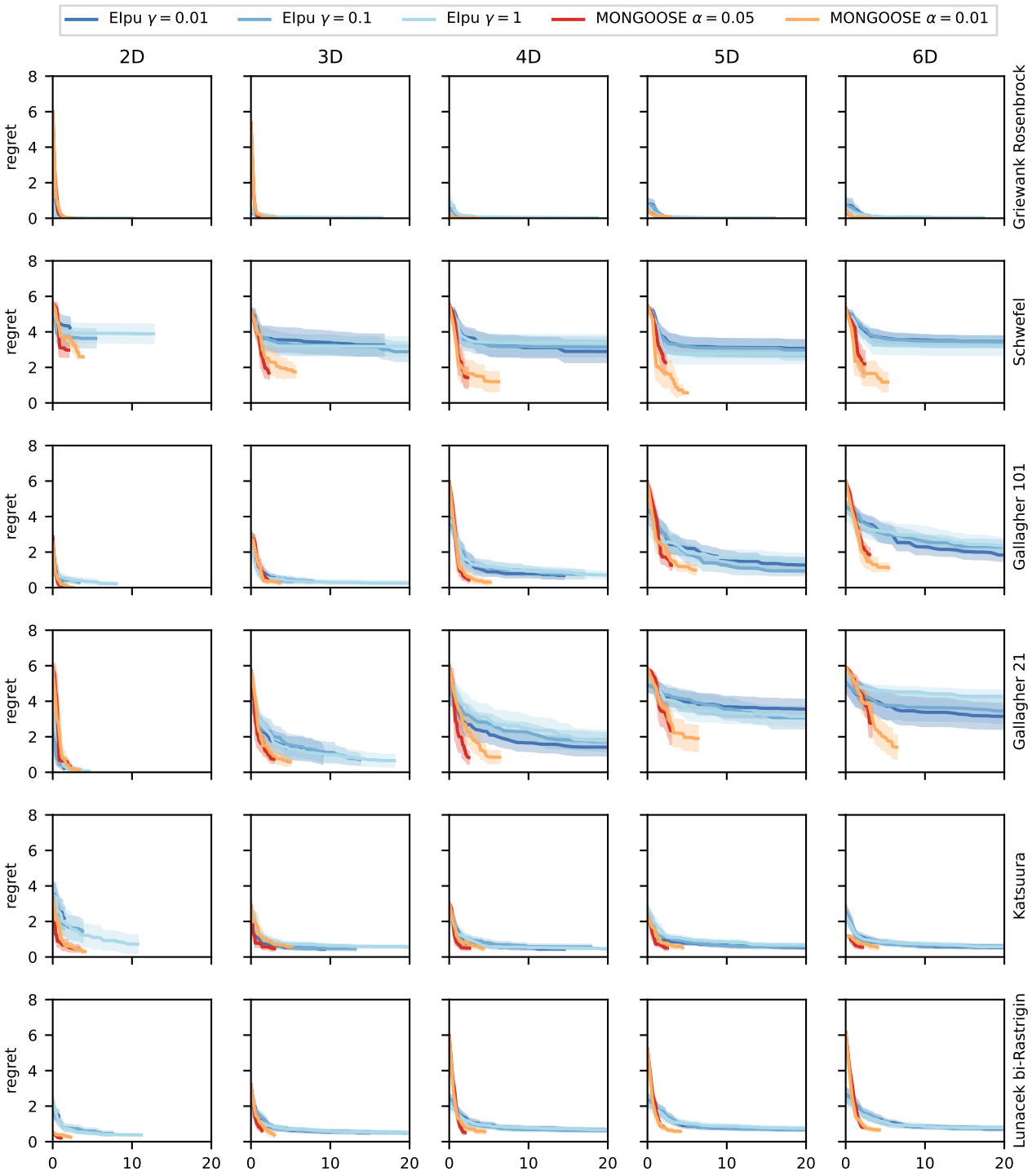


Figure 48: Individual COCO plots for Figure 19. COCO functions 19-24: composite Griewank-Rosenbrock function, Schwefel function, Gallagher's Gaussian 101-me peaks function, Gallagher's Gaussian 21-hi peaks function, Kastsuura function, Lunacek bi-Rastrigin function.

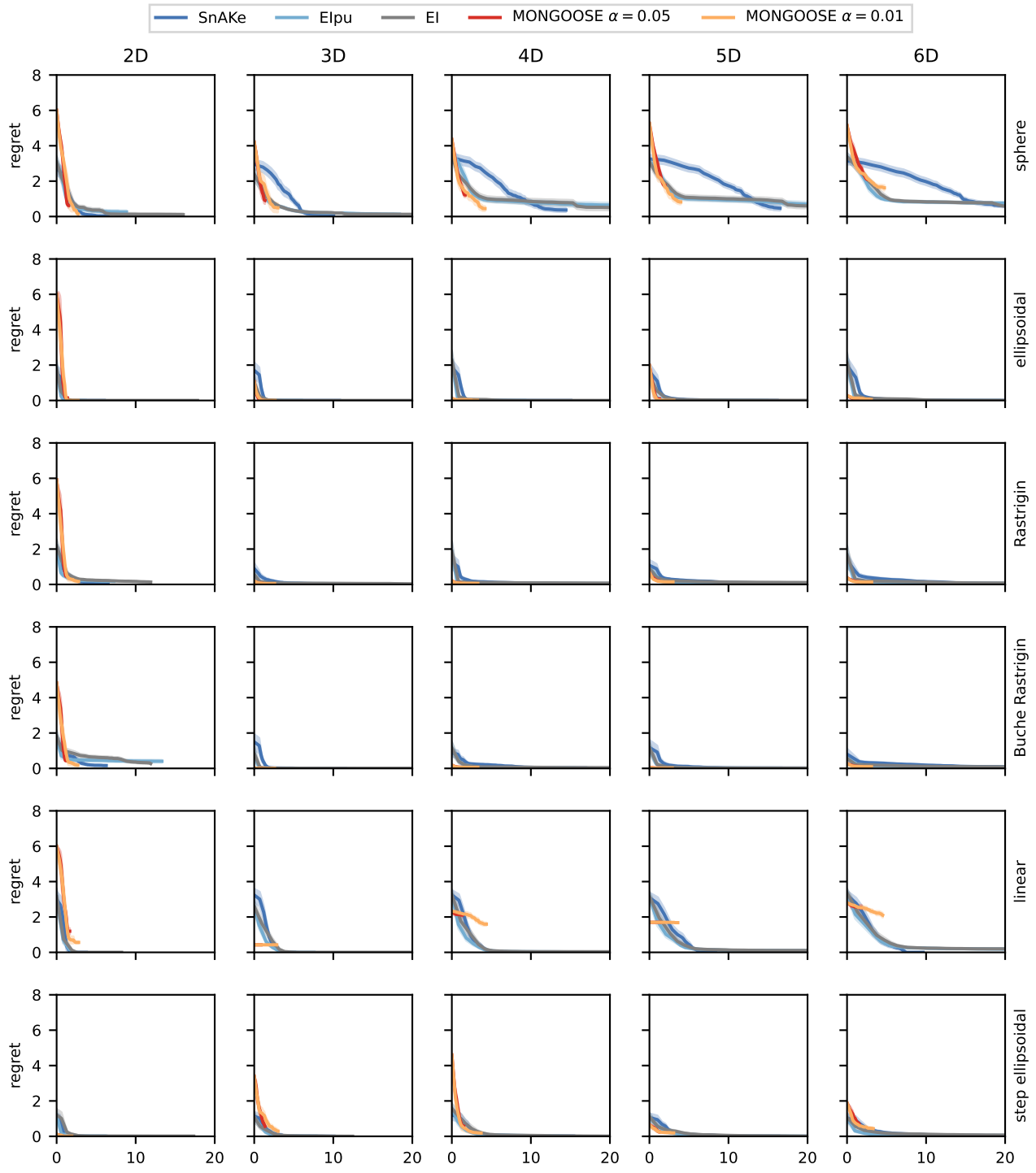


Figure 49: Individual COCO plots for Figure 4. COCO functions 1-6: sphere function, ellipsoidal function, Rastrigin function, Büche-Rastrigin function, linear slope, step ellipsoidal function.

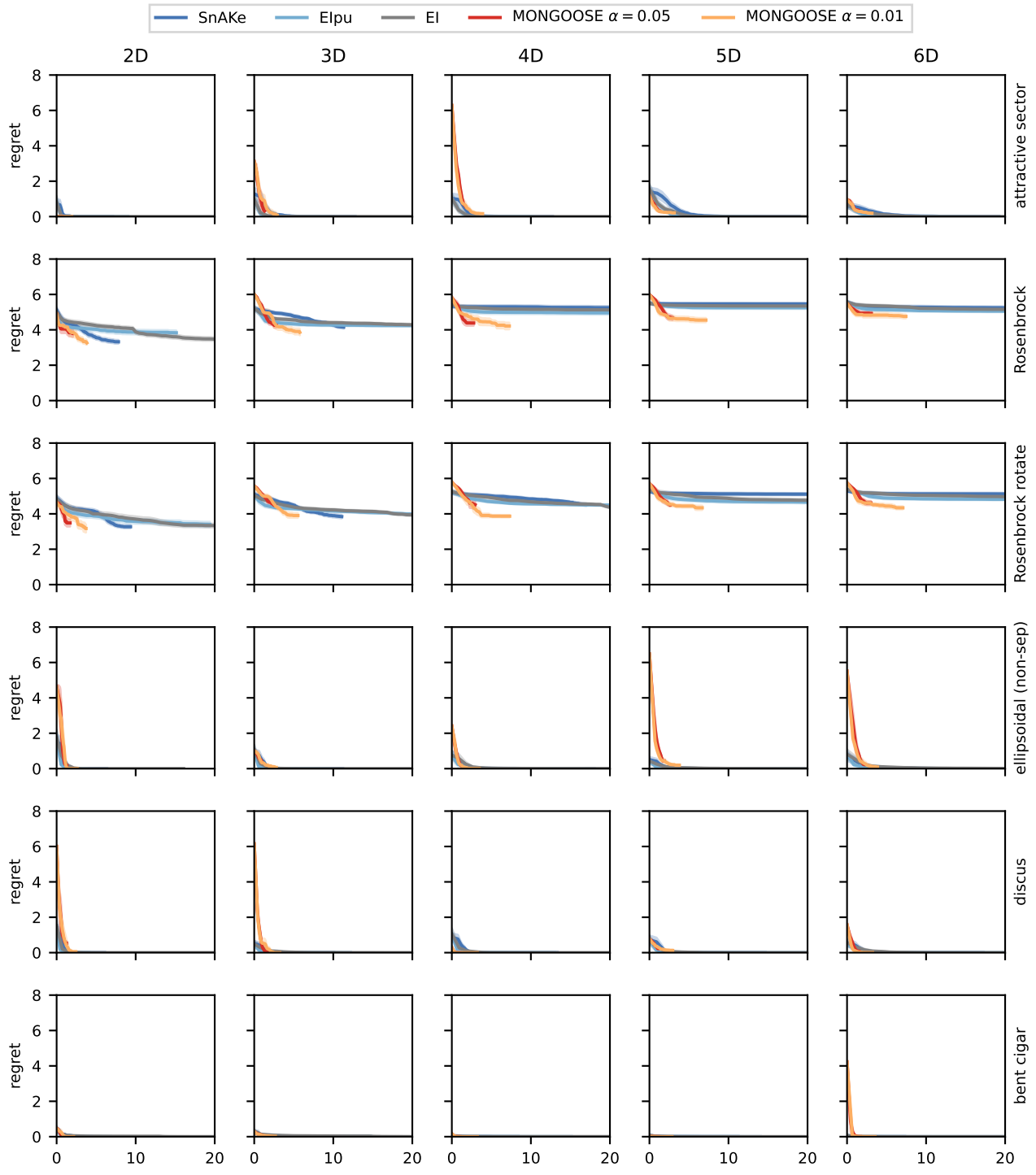


Figure 50: Individual COCO plots for Figure 4. COCO functions 7-12: attractive sector function, Rosenbrock (original) function, Rosenbrock (rotated) function, ellipsoidal (non-separable) function, discus function, bent cigar function.

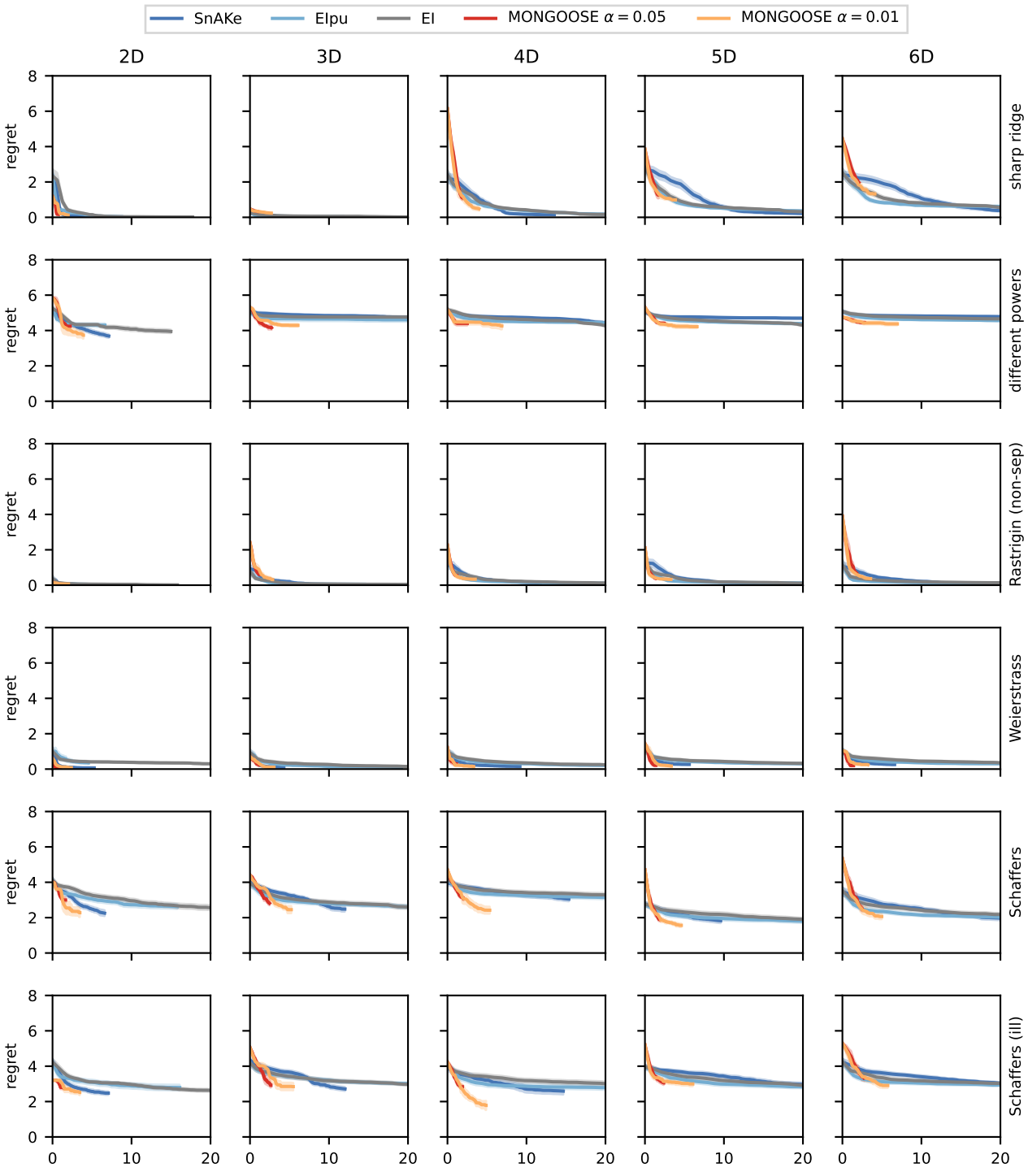


Figure 51: Individual COCO plots for Figure 4. COCO functions 13-18: sharp ridge function, different powers function, Rastrigin (non-separable) function, Weierstrrass function, Schaffers F7 function, Schaffers F7 (moderately ill-conditioned) function.

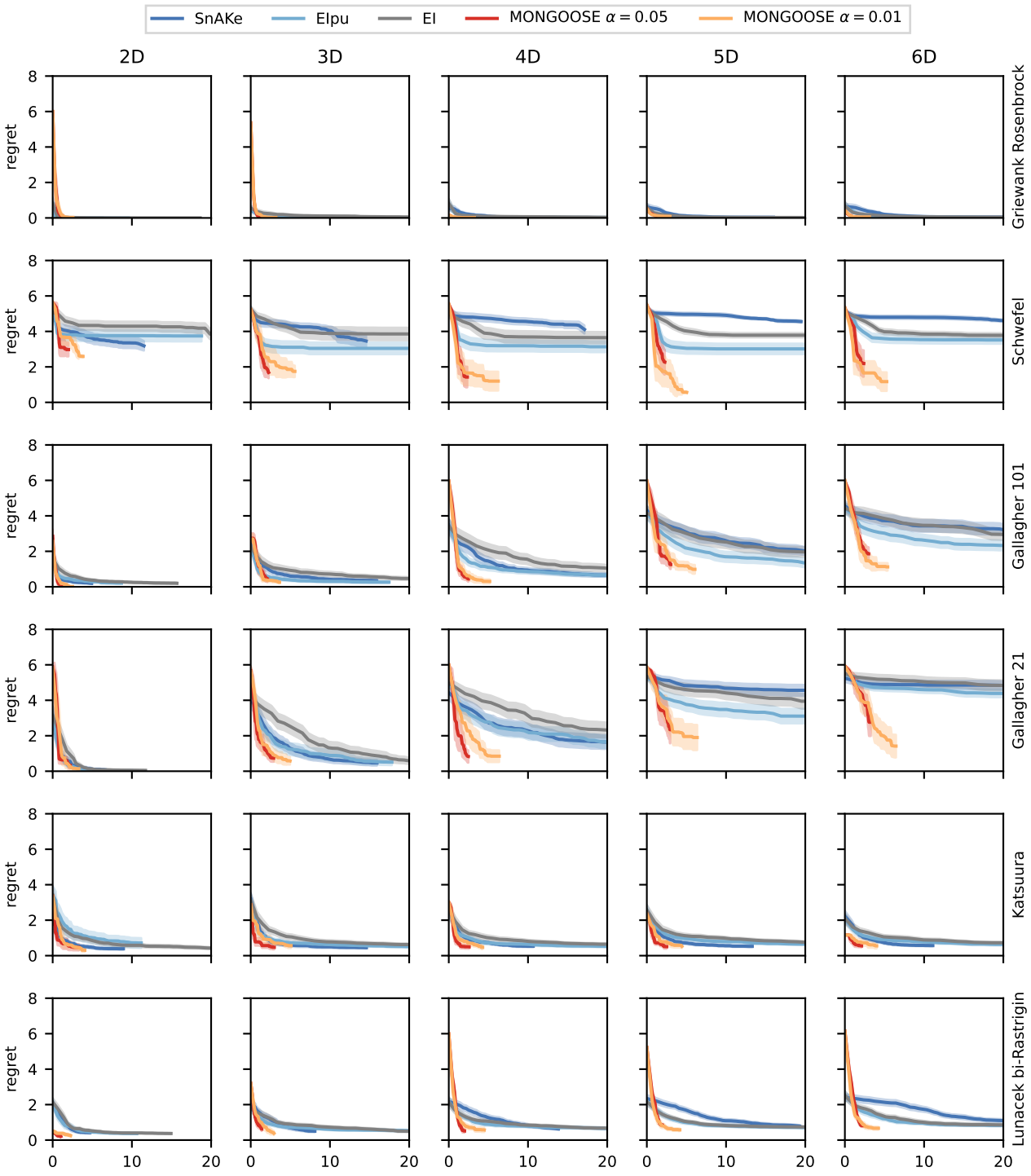


Figure 52: Individual COCO plots for Figure 4. COCO functions 19-24: composite Griewank-Rosenbrock function, Schwefel function, Gallagher's Gaussian 101-me peaks function, Gallagher's Gaussian 21-hi peaks function, Kastsuura function, Lunacek bi-Rastrigin function.