Self-Adjusting Weighted Expected Improvement for Bayesian Optimization

Anonymous[1]

[1]Anonymous Institution

Abstract Bayesian Optimization (BO) is a class of surrogate-based, sample-efficient algorithms for optimizing black-box problems with small evaluation budgets. The BO pipeline itself is highly configurable with many different design choices regarding the initial design, surrogate model, and acquisition function (AF). Unfortunately, our understanding of how to select suitable components for a problem at hand is very limited. In this work, we focus on the definition of the AF, whose main purpose is to balance the trade-off between exploring regions with high uncertainty and those with high promise for good solutions. We propose Self-Adjusting Weighted Expected Improvement (SAWEI), where we let the exploration-exploitation trade-off self-adjust in a data-driven manner, based on a convergence criterion for BO. On the BBOB functions of the COCO benchmarking platform, our method exhibits a favorable any-time performance compared to handcrafted baselines and serves as a robust default choice for any problem structure. The suitability of our method also transfers to HPOBench. With SAWEI, we are a step closer to on-the-fly, data-driven, and robust BO designs that automatically adjust their sampling behavior to the problem at hand.

1 Introduction

Black-box problems are challenging to optimize because we do not have direct access to the underlying structure of the problem landscape. To optimize them, we can sequentially evaluate different points x and use the obtained objective values $f(x)$ to choose which point(s) to evaluate next, but we do not have *a priori* information where to find the most promising regions or how to best trade off exploration of the search space with exploitation of regions that appear to be very promising. Formally, in black-box optimization we want to find the minimum x^* of a given function f, $x^* \in \arg\min_{x \in \mathcal{X}} f(x)$, without having access to the function itself other than through the queries. Typical black-box problems occur in engineering or hyperparameter optimization (HPO), where the quality of potential solutions is evaluated via numeric simulations or training machine learning models.

Balancing exploration with exploitation is particularly challenging when we have a low number of available function evaluations in relation to the size of the search space $\mathcal{X}$. A popular approach to address such settings is Bayesian optimization (BO) (Mockus, 1989; Garnett, 2023), often promoted as sample-efficient for expensive black-box optimization. The main idea of BO is to use a probabilistic surrogate model (e.g., a Gaussian Process), iteratively refining an approximation of the problem landscape that guides the optimization process. BO starts with an *initial design* or *design of experiment* (DoE), obtained from sampling strategies, e.g., random sampling, Sobol sequence or Latin Hypercube desing (Brochu et al., 2010). With these initial points, the surrogate model is built to capture the uncertainty of the true cost on unobserved points. The *acquisition function* (AF) (a.k.a. infill criterion) is a utility function to trade off exploration of underexplored areas and exploitation of presumably promising areas. The point with the highest acquisition function value is queried next, the surrogate model is adjusted with the new observation, and the optimum is updated if the new point improves the target value of the best-so-far observation. These steps are repeated for a given overall optimization budget.

 © 2023 the authors, released under CC BY 4.0

Besides accurate probabilistic surrogate models and other bells and whistles (Lindauer et al., 2019; Bossek et al., 2020; Cowen-Rivers et al., 2021), the exploration-exploitation trade-off is crucial for successful and efficient optimization. Since the landscape of the black-box optimization problem is unknown, it is a priori unclear which AF should be chosen for the optimization problem at hand. Even worse, since each problem has its unique landscape, we need different exploration-exploitation trade-offs (Benjamins et al., 2022a,b).

Because there are different choices of AFs, e.g., Probability of Improvement (PI) (Kushner, 1964), Expected Improvement (EI) (Mockus et al., 1978), Upper Confidence Bound (UCB) (Forrester et al., 2008), Thompson Sampling (TS) (Thompson, 1933), Entropy Search (Hennig and Schuler, 2012) and Knowledge Gradient (Frazier et al., 2009), selecting a suitable one for the problem at hand with insights on the landscape remains challenging. Furthermore, in the past, the choice of an AF has been considered *static* over the BO process. Prior works suggest that mixed AF-strategies (Hoffman et al., 2014; Kandasamy et al., 2020) or even very simple schedules switching from EI to PI can improve anytime performance of BO; however, for each problem different schedules, incl. static ones, perform best (Benjamins et al., 2022b).

Performance can be improved by selecting an AF-schedule with a meta-learned selector based on the exploratory landscape analysis (ELA) features (Mersmann et al., 2011) of the initial design which factors in the problem at hand (Benjamins et al., 2022a). Nevertheless, this approach has its limitations. First, it requires a large and expensive initial design compared to the overall budget in order to compute the ELA features, and the ideal size of it is unknown. Second, the selector is trained for a specific budget, and it is unclear how it transfers to other dimensions, optimization budgets or initial designs.

In this work, we instead aim for a *self-adjusting yet simple* approach to adapt the exploration-exploitation trade-off in a data-driven way throughout the optimization process. For this, we propose to adaptively set the weight α of Weighted Expected Improvement (WEI) (Sobester et al., 2005) in an online control fashion. Depending on how we parametrize WEI we can be fully explorative, recover EI or lean towards a modulated, exploitative PI. The crucial questions to answer here are (i) *When* should we adjust α? and (ii) *how* should we adjust α?

We propose a new method, dubbed Self-Adjusting Weighted Expected Improvement (SAWEI). Inspired by a termination criterion for BO (Makarova et al., 2022), we adjust the weight α whenever BO tends to converge, indicated by the Upper Bound Regret (UBR). We adjust α opposite to the dominant search attitude, either towards exploration or exploitation. The key mechanism behind SAWEI is illustrated in Figure 1. We demonstrate the effectiveness of our method SAWEI on the BBOB functions of the COCO benchmark (Hansen et al., 2020) and on tabular benchmarks from HPOBench (Eggensperger et al., 2021) against baselines of established AFs and previously proposed handcrafted AF-schedules for α.

2 Related Work

We first present existing work on AFs and report findings about methods focusing on the improvement and development of better AFs, from enhancing EI (Qin et al., 2017; Balandat et al., 2020) to meta-learning neural AFs (Volpp et al., 2020). A different line of work is concerned with combining different AFs, e.g. by building a portfolio of AFs (EI, PI, UCB with different hyperparameter settings) and then using an online multi-armed bandit strategy to assign probabilities of which AF to use at which step, called GP-Hedge or Portfolio Allocation (Hoffman et al., 2014). Their work indicates that the performance of GP-Hedge highly varies with the number of arms and their respective hyperparameter settings. Similarly to Portfolio Allocation, Kandasamy et al. (2020) update weights of their portfolio (UCB, EI, TS (Thompson, 1933), Top-Two Expected Improvement (TTEI) (Qin et al., 2017)) in an online manner. They do not include PI as they observe it exhibits inferior performance compared to other single static AFs. In addition, robust versions of EI, PI and UCB can be combined to a multi-objective AF combining the strengths of the individual ones (Cowen-Rivers et al., 2021).

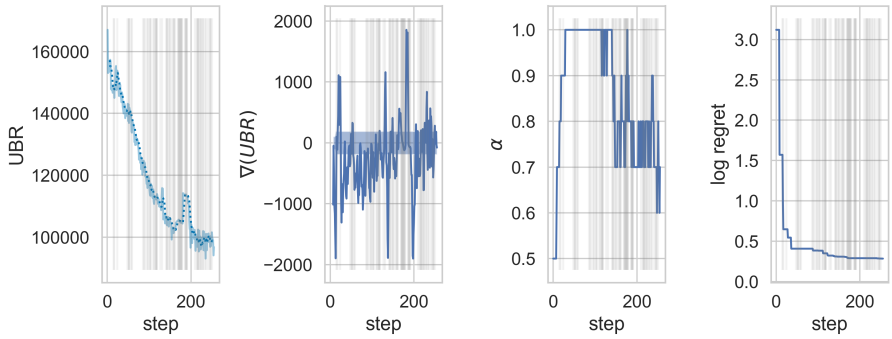


Figure 1: With SAWEI we self-adjust the exploration-exploitation trade-off parameter α based on the Upper Bound Regret (UBR) (left). Whenever the gradient of UBR (2nd left) becomes 0 (marked by vertical lines), we adjust α (2nd right), further reducing the log regret (right). BBOB F20, 8d.

In this work, we take a step back and ask ourselves what we could achieve by employing a simplistic approach of self-adjusting the exploration-exploitation trade-off of WEI.

It has also been shown in other optimization-related areas that dynamic choices are beneficial in terms of performance, e.g. in evolutionary computation Karafotias et al. (2015); Doerr and Doerr (2020), planning (Speck et al., 2021) and deep learning (Adriaensen et al., 2022). Recently, the introduction of Dynamic Algorithm Configuration (DAC) (Biedenkapp et al., 2020) underlines the potential of employing dynamic schedules (as opposed to selecting algorithm components *on the fly*, as is usually done in evolutionary computation (Hansen et al., 2003)).

Related to that, also setting the weight α of WEI has been investigated. Sobester et al. (2005) propose to cycle through $\alpha \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ to pulse from exploring to exploiting. This idea is based on the suggestion to cycle through global-local balances during the search (Gutmann, 2001). However, this heuristic is oblivious to the current state of the search. Another line of work proposes to simply query WEI n times with n different values of α in parallel (Liu et al., 2018) with the drawback of potentially uninformative function evaluations. The weights for the exploration and exploration terms in WEI can also be set via rewards obtained by calculating the accuracy of the surrogate model (Xiao et al., 2012, 2013). However, the definition of the rewards is lacking and their method needs to be reset from time to time for the case when the exploration term causes to repeatedly propose the same configuration.

3 Self-Adjusting Weighted EI

In our method, the Self-Adjusting Weighted Expected Improvement (SAWEI), we adaptively set the weight $\alpha \in [0, 1]$ of the Weighted Expected Improvement (WEI) to steer the exploration-exploitation trade-off. WEI (Sobester et al., 2005) is defined as:

$$WEI(\mathbf{x}; \alpha) = \alpha \underbrace{z(\mathbf{x})\hat{s}(\mathbf{x})\Phi\left[z(\mathbf{x})\right]}_{\text{exploitation}} + (1-\alpha) \underbrace{\hat{s}(\mathbf{x})\phi\left[z(\mathbf{x})\right]}_{\text{exploration}} \tag{1}$$

with $z(\mathbf{x}) = (f_{\min} - \hat{y}(\mathbf{x}))/\hat{s}(\mathbf{x})$, $f_{\min}$ being the lowest observed function value, $\hat{y}(\mathbf{x})$ and $\hat{s}(\mathbf{x})$ the predicted mean and standard deviation from the surrogate model, and ϕ and Φ being the PDF and CDF of a Gaussian distribution, respectively. The α coefficient weighs the exploration and exploitation terms. For example, $\alpha = 0.5$ recovers standard EI (Mockus et al., 1978) and $\alpha = 1$ has a similar behavior as $PI(\mathbf{x}) = \Phi\left[z(\mathbf{x})\right]$ (Kushner, 1964).

When To Adjust. In order to be able to set α adaptively, we need an indicator of the progress of the optimization. Recently, Makarova et al. (2022) proposed a termination criterion to stop BO for hyperparameter optimization. If the Upper Bound Regret (UBR) falls under a certain threshold, they terminate. UBR estimates the true regret at iteration k by:

$$\mathrm{UBR}(G_k; \mathcal{X}) = r_k := \min_{\mathbf{x} \in G_k} \mathrm{UCB}_k(\mathbf{x}) - \min_{\mathbf{x} \in \mathcal{X}} \mathrm{LCB}_k(\mathbf{x}) \tag{2}$$

with G_k being the history of all evaluated points, $\mathcal{X}$ being the entire search space, and LCB and UCB being the lower and upper confidence bound, e.g., $UCB(x) = \mu_t(x) + \beta_t \sigma_t(x)$ and $LCB(x) = \mu_t(x) - \beta_t \sigma_t(x)$, respectively. The first term of UBR estimates the worst-case function value of the best-observed point, a.k.a. the incumbent, and the second term is the lowest function value across the whole search space. This means the smaller the gap between both terms becomes, the closer we are at the asymptotic function value *under the current settings of the optimizer*. The UBR does not directly operate on function values, but UCB and LCB are computed on the surrogate model instead. Instead of using UBR to stop the optimization process, it serves as an indicator for us when to switch or adjust components.
Our rule is: When the gradient of UBR over the last n steps becomes close to 0, we adjust the exploration-exploitation attitude with α. The sensitivity to the gradient is controlled by our hyperparameter ϵ. Please find the details in Appendix A.

How to Adjust. The remaining question is *how* to adjust α, by how much and into which direction. We propose a rather simple, yet effective additive change by Δ_α. We set $\Delta_\alpha = 0.1$ to allow for gradual changes. We determine the sign of Δ_α by the recent optimization *attitude*: Depending on whether the exploration- or exploitation-term of Equation (1) is larger for the last selected point, the current search attitude was either steered more for exploring or exploiting, resp. For a clearer signal, instead of using $z(\mathbf{x})\hat{s}(\mathbf{x})\Phi\left[z(\mathbf{x})\right]$ we use a modified exploitation term, that is $\Phi\left[z(\mathbf{x})\right]$ which is equal to PI. We therefore compare the exploration term $\hat{s}(\mathbf{x})\phi\left[z(\mathbf{x})\right]$ to the modified exploitation term $\Phi\left[z(\mathbf{x})\right]$. We inspect the attitude and adjust α in the *opposite* direction, to provide a chance for more exploration or exploitation in contrast to the currently dominating attitude.

SAWEI in a Nutshell. We illustrate and summarize our method SAWEI in Figure 1 and in Algorithm 1. Our goal is to adjust the exploration-exploitation trade-off based on the current search attitude whenever the Upper Bound Regret (UBR) converges. SAWEI enhances the standard BO pipeline by calculating the UBR in each iteration and by tracking the search attitude via the exploration term and the exploitation term of WEI. First, we define and evaluate the initial design and train our surrogate model (Line 1). Then, as long as we have function evaluations left (Line 2), we query the acquisition function (here Weighted Expected Improvement (WEI)) for the next point to be evaluated (Line 3). Meanwhile, we track the search attitude with the exploration and exploitation terms of WEI (Line 4). The function is evaluated as usual with the proposed point and we update our history and our surrogate model (Lines 5-7). Now we calculate the UBR estimating the gap to the true regret based on the history of evaluated points and the search space (Line 8). We smooth the history of UBR with moving interquartile mean (IQM) (Lines 9-10). Based on this smoothed version, we check whether UBR has converged, i.e., the gradient of UBR is close to 0 (Line 11). If it is the case, we adjust the weight α of WEI based on the search attitude (Line 12). The search attitude is calculated with the exploration and exploitation terms of WEI.

4 Experiments

In our experiments, we empirically evaluate our method SAWEI on different benchmarks and compare it to baselines from the literature and handcrafted ones. We benchmark the schedules on the BBOB functions from the COCO problem suite (Hansen et al., 2020) and on

Algorithm 1 Bayesian Optimization with Self-Adjusting Weighted Expected Improvement (SAWEI)

Require: Initial weight of WEI $\alpha = 0.5$, history of evaluated points $G = \emptyset$, history of regret estimates/UBR R, surrogate model $\mathcal{M}$, function to optimize f

Evaluate initial design and train surrogate model $\mathcal{M}$
while Optimization Budget Not Exhausted **do**
  $x_{\text{next}} \leftarrow \text{WEI}(\mathcal{M})$ ▷ Propose next configuration to evaluate
  $a_{\text{explore}}, a_{\text{exploit}} \leftarrow \text{WEI}(\mathcal{M})$ ▷ Get summands of WEI before the surr. model is trained
  $y \leftarrow f(x_{\text{next}})$ ▷ Evaluate function
  $G \leftarrow G \cup \{x_{\text{next}}\}$ ▷ Update history
  Train surrogate model $\mathcal{M}$
  $r \leftarrow \text{UBR}(G, \mathcal{X})$ ▷ Upper Bound Regret (UBR) estimate, Equation (2)
  $R \leftarrow R.\texttt{append}(r)$
  $\bar{R} \leftarrow \texttt{smooth_with_iqm}(R)$ ▷ Smooth rugged signal with moving IQM
  if $\nabla \bar{R} \approx 0$ **then** ▷ Check if UBR converged
    $\alpha \leftarrow \texttt{adjust}(\alpha, a_{\text{explore}}, a_{\text{exploit}})$ ▷ Adjust exploration-exploitation based on attitude
  end if
  end while

HPOBench (Eggensperger et al., 2021). Our implementations are built upon the BO tool SMAC3 (v2.0.0b1) (Lindauer et al., 2022). The code is available at `https://anon-github.automl.cc/r/SAWEI-9CD6/`. The exact setting for our method is $\epsilon = 0.1$ and adding or subtracting $\Delta\alpha = 0.1$. We set our convergence check horizon to $n = 1$, i.e., we check whether the last gradient is close to 0. We validate our hand-crafted settings in the ablation in Section 4.1.

Our evaluation protocol repeats the optimization 10 times with different random seeds and calculates the interquartile mean (IQM) across seeds to robustly estimate the regret per function. For each schedule, we then determine the rank for each of the 24 BBOB functions and compute the global rank across functions. For the rank table, we aggregate the ranks across the single tasks per schedule with the IQM. In the plots over optimization steps, we show the mean and 95 % confidence interval across all the functions.

BBOB. For the 24 noiseless, synthetic BBOB functions (Hansen et al., 2020) we set the dimensionality to 8, the budget of the initial design to 24 function evaluations (FEs), and the budget for the surrogate-based optimization to 256 FEs. We optimize the first three instances of each function. In BBOB, the instances are obtained by scaling, shifting, and rotating the base function (hence preserving the problem structure but changing the embedding).

HPOBench. We evaluate all methods on the tabular machine learning benchmarks from HPOBench (Eggensperger et al., 2021). To this end, we randomly selected five tasks from the OpenML dataset (Casalicchio et al., 2017; Feurer et al., 2021) and optimize a Random Forest, MLP, SVM, Logistic Regression and XGBoost. We allow an initial design of 15 FEs and a BO-based optimization budget of 100 FEs. For each FE, we average the metric over the five available seeds.

Baselines. We compare our data-driven, self-adjusting method SAWEI to (i) the well-established best practice of simply using a single AF and (ii) hand-designed schedules of α, see Table 1. We start with static schedules of $\alpha \in \{0, 0.5, 1\}$, either fully exploring, EI, or a modulated PI.[1] Further, we define a schedule from EI to modulated PI (and vice versa) as a step function with 5 steps. In addition, we compare to hard switches from EI to PI (Benjamins et al., 2022a) as well as the Gutmann-Sobester pulse cycling through α (Gutmann, 2001; Sobester et al., 2005). We also include

[1] Following Eq. (1), setting $\alpha = 1$ results in a modulated PI and not the original PI.

Explore	$\alpha = 0.0$
EI	$\alpha = 0.5$
PI*	$\alpha = 1.0$
EI $\rightarrow$ PI* (Linear)	5 steps
PI* $\rightarrow$ EI (Linear)	5 steps
EI $\rightarrow$ PI	switch after 25 %
EI $\rightarrow$ PI	switch after 50 %
EI $\rightarrow$ PI	switch after 75 %
Gutmann-Sobester Pulse (Gutmann, 2001; Sobester et al., 2005)	Cycle $\alpha \in [0.1, 0.3, 0.5, 0.7, 0.9]$
Portfolio Allocation (Hoffman et al., 2014)	-

Table 1: Baselines. PI* denotes WEI($\alpha = 1$).

Portfolio Allocation (Hoffman et al., 2014) and use their portfolio of nine acquisition functions consisting of different parametrizations of UCB, PI and EI.

4.1 Results

BBOB. Our method SAWEI ranks among the first based on final performance (cf. Figure 5), which is very similar to dynamic baselines going from EI ($\alpha = 0.5$) to the modulated PI ($\alpha = 1$). One drawback of the hand-designed schedules is that the optimization budget needs to be defined beforehand, whereas our method is self-adjusting and is oblivious of the total budget. Surprisingly, the modulated PI is comparatively strong and performs better than EI, suggesting that the BBOB landscapes require a higher percentage of exploitation. SAWEI also exhibits a favorable any-time performance, making it a consistent and robust default choice, see Figure 2. Schedules dominating SAWEI only do so for a portion of the optimization, hence they are not consistent. Confirming results from Benjamins et al. (2022b), the effect of switching from EI to PI can be clearly seen as a boost in the ranks. On BBOB, the general good performing schedules involve PI which our method can easily mimic. SAWEI finds a suitable transition from exploring to exploiting per-run.

In general, the tendency of the α-schedules traversed by SAWEI is moving from exploration to exploitation. Often, we can observe a decrease, a change to more exploration again, after some iterations. On an exemplary BBOB function, the multi-modal Schwefel function (F20) with weak global structure (Figure 3a), SAWEI manages to efficiently transform from exploring to exploiting. At the end of the optimization, when the basin was already discovered, SAWEI decreases α to more exploration to explore the surroundings. We can also clearly observe the effect of the hand-designed switching here, although SAWEI discovers a more suitable point and can change its attitude again.

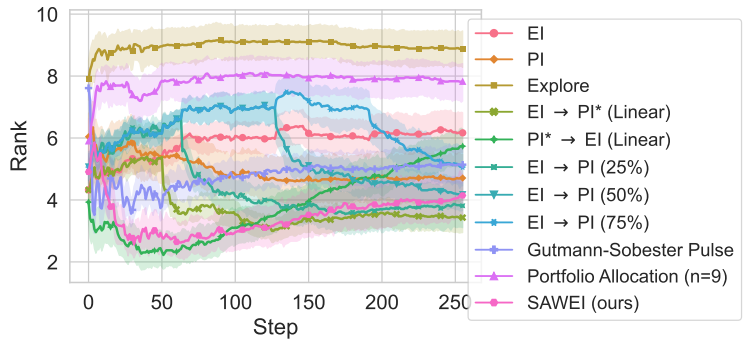
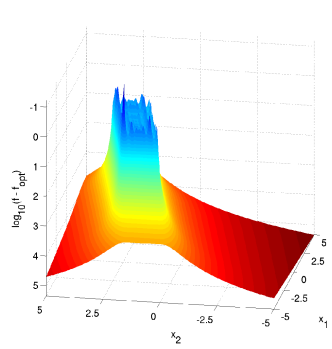
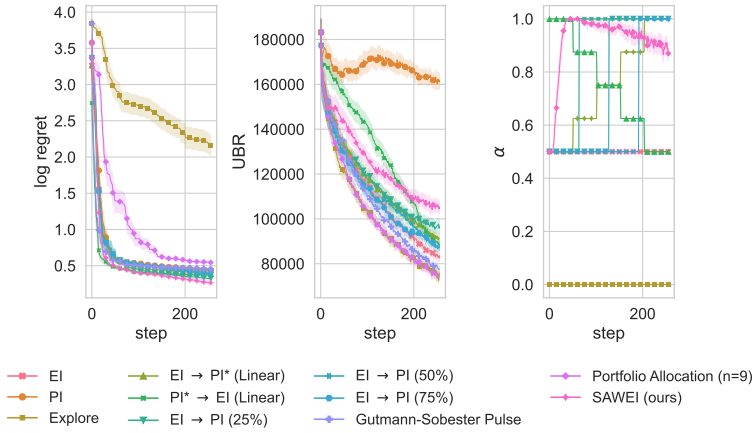


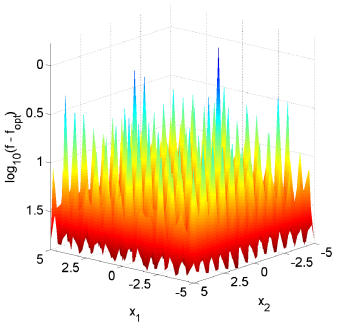
Figure 2: Ranks per Step on BBOB



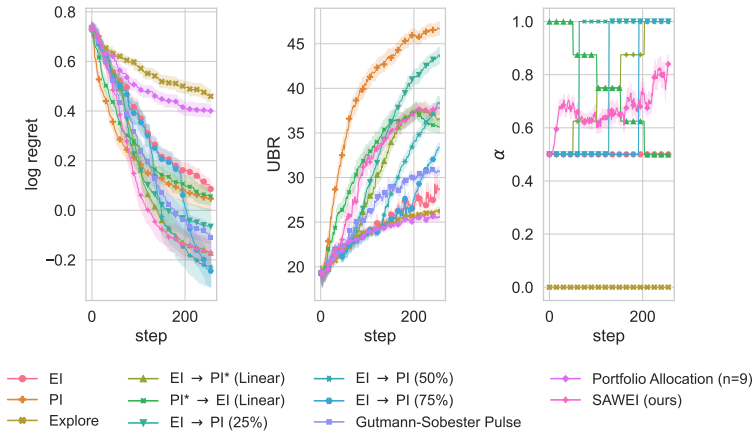
(a) BBOB F20: Schwefel Function (Hansen et al., 2020) (Image source (Finck et al., 2009))



(b) F20: Log regret, UBR and α-schedules



(c) BBOB F23: Katsuura Function (Hansen et al., 2020) (Image source (Finck et al., 2009))



(d) F23: Log regret, UBR and α-schedules

Figure 3: Exemplary BBOB Functions F20 and F23 with different performance and behavior indicators

On Katsuura (highly multi-modal, weak global structure, F23) SAWEI α is increased much more slowly to exploitation, presumably because of the highly rugged landscape. Also here, SAWEI discovers the boost from changing from exploration to exploitation. If we look closely we can see that the Upper Bound Regret jumps up after the switch happened for the switching schedules (EI to PI) which is an indication of the adequacy of UBR as a state descriptor. Please find all schedule plots for each BBOB function as well as the box plots of the final log regrets in Appendix C.

HPOBench. On HPOBench we see that SAWEI also has a favorable any-time performance, see Figure 4, and ranks among the first for the final log regret (Figure 5). It is on par with Explore ($\alpha = 0$), and they are directly followed by Portfolio Allocation and EI. The supremacy of the exploratory schedules is quite surprising, given the simplicity commonly attributed to response landscapes in HPO (Pushak and Hoos, 2018). We will investigate this further in our future work. With a closer look at the schedules, we see the general trend to start from EI ($\alpha = 0.5$) and go to Explore ($\alpha = 0$) which is the complete opposite of the BBOB behavior. Please find the boxplot of the final log regret and all performance over time with α in Appendix D.

In summary, we observe that the optimal schedule and search behavior vary on two levels. First, for a given problem type, the optimal schedule varies across the single tasks. Second, the search behavior depends on the type of problem, whether we optimize synthetic functions in BBOB

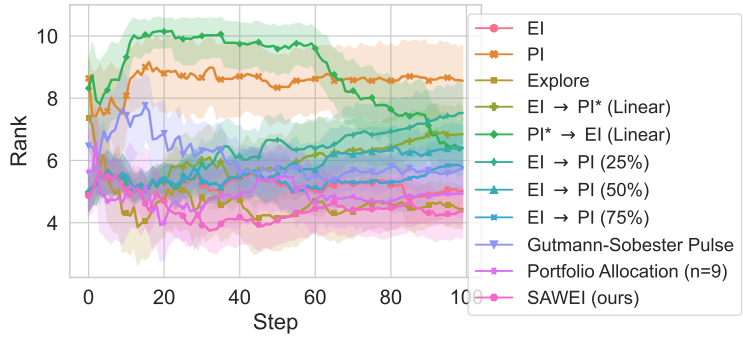


Figure 4: Ranks per Step on HPOBench

or find optimal hyperparameters for machine learning models in HPOBench. SAWEI mimics the strategy fitting best to the problem at hand and exhibits the most favorable rank distribution *across domains*, see Figure 5.

Ablation on BBOB. Because our method itself has hyperparameters, we perform an ablation study to find a well-performing version. In particular, we vary $\Delta\alpha \in \{0.05, 0.1, 0.25\}$, i.e., the amount to add or subtract to our current weight α. In addition, we can track the attitude in different ways: Either just considering the last step, or accumulating the terms until the last incumbent change or until the last switch. In the latter cases a_{explore} and a_{exploit} become sums. This hyperparameter defines the convergence check horizon n, which is varied during the run for the latter two options. Finally, we vary the sensitivity to the gradient of UBR by the width of the tolerance band when compared to 0: $\epsilon \in \{0.05, 0.1, 0.5, 1\}$. The bigger ϵ the more often we switch. We assess the hyperparameter importance with fANOVA (Hutter et al., 2014) on all BBOB functions with 10 seeds and 1 instance on 8 dimensions. We normalize the log regret for each BBOB function and use this as the performance metric.

We show the marginals of each hyperparameter in Figure 6. The sensitivity ϵ to the gradient has a slight tendency to 0.1 but the overall differences are small, we argue that the exact timing of the signal to adjust is less important. In addition, setting the granularity of $\Delta\alpha$ is quite robust to the exact setting. In contrast, tracking the attitude has a tendency to favor checking the exploration/exploration terms until the last incumbent change. It is likely that on other benchmarks

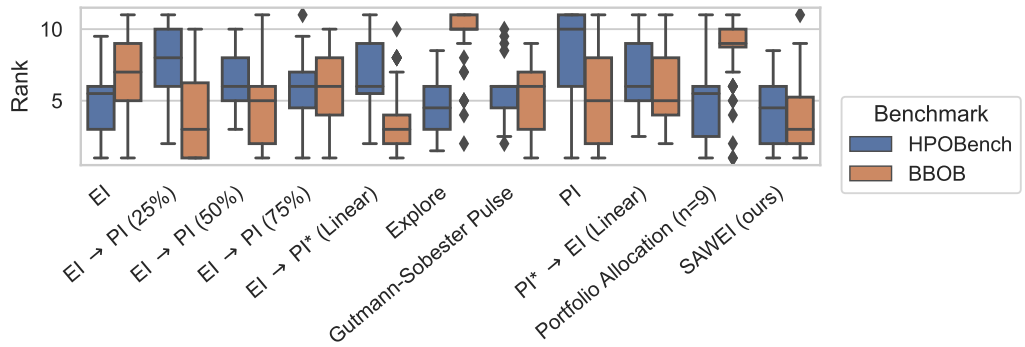


Figure 5: Our method SAWEI has the most favorable rank distribution *across domains*. That is, even though on different benchmarks other schedules perform on par, their suitability highly differs depending on the benchmark.

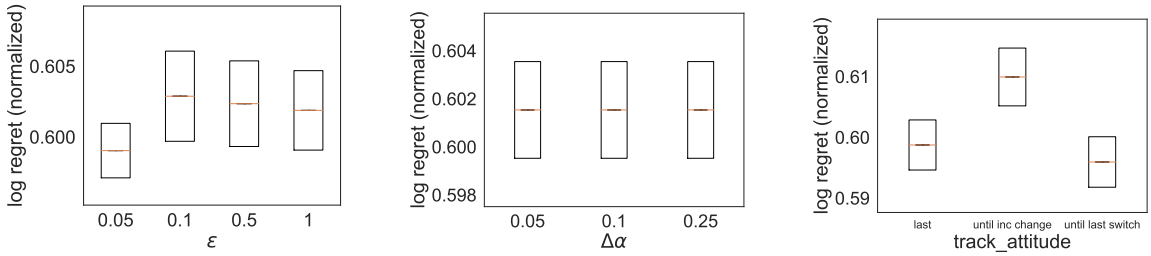


Figure 6: SAWEI Hyperparameter Importance (the higher the better).

the importances might change and our default of $\epsilon = 0.1$, `track_attitude` = last, $\Delta\alpha = 0.1$ proves to be a robust one.

5 Limitations and Future Work

Our method SAWEI introduces a slight overhead due to the need to optimize Lower Confidence Bound (LCB) for computing UBR in each iteration. Everything that follows, namely deciding whether and how to adjust α, is negligible in terms of computational cost.

In our analysis, we did not experiment with the initial value of α, which may not be optimal for every tested function. Also, our method does not allow jumps or resetting α, which could also be beneficial. In this context, defining α directly as a function of the exploration/exploitation terms of WEI could be a way to allow more flexibility.

One limitation is that so far we have only combined EI and PI. Our approach can easily be extended to *any* linear combination of two acquisition functions. Moreover, we can combine SAWEI with Dynamic Algorithm Configuration (DAC) (Biedenkapp et al., 2020) to learn policies of α across instances and tasks. More generally, we strongly believe that meta-learning and self-adjustment should go hand in hand, another topic to be explored in future work. Building on (Benjamins et al., 2022a), one could consider to warmstart SAWEI using meta-models utilizing ELA features (Mersmann et al., 2011). Finally, we believe that also other components of BO could benefit from self-adjusting choices.

6 Conclusions

Through a self-adjusting choice of the acquisition function in Bayesian Optimzation we aim to benefit from two main levers: (1) an automated identification of the AF best suitable for the unknown task at hand (e.g., while PI performs better than EI on BBOB, it is the other way around for HPO problems), and (2) an adjustment to the different needs during the optimization process.

Our method SAWEI uses the convergence of Upper Bound Regret (UBR) as a criterion when to adjust its parametrized *acquisition function* (AF). SAWEI proves to achieve promising performance on two classic benchmark suites, outperforming the static EI and PI AFs. It is hence able to achieve both goals, (1) and (2), listed above. It furthermore does not only achieve good final ranks, but also exhibits a favorable any-time performance on both suites, BBOB and HPOBench.

As a side result of our study, we observe that the general trends in BBOB and HPOBench are orthogonal to each other: while SAWEI generally traverses from EI (exploration) to a modulated PI (exploitation) for BBOB, it moves from EI to full exploration on HPOBench. This demonstrates the need for flexible, on-the-fly-adjustment of BO components.

Broader Impact Statement: After careful reflection, the authors have determined that this work presents no notable negative impacts on society or the environment.

References

Adriaensen, S., Biedenkapp, A., Shala, G., Awad, N., Eimer, T., Lindauer, M., and Hutter, F. (2022). Automated dynamic algorithm configuration. *arXiv:2205.13881 [cs.AI]*.

Balandat, M., Karrer, B., Jiang, D., Daulton, S., Letham, B., Wilson, A., and Bakshy, E. (2020). Botorch: A framework for efficient monte-carlo Bayesian optimization. In *Proc. of NeurIPS'20*.

Benjamins, C., Jankovic, A., Raponi, E., van der Blom, K., Lindauer, M., and Doerr, C. (2022a). Towards automated design of Bayesian Optimization via Exploratory Landscape Analysis. In *Meta-Learning Workshop @ NeurIPS'22*.

Benjamins, C., Raponi, E., Jankovic, A., van der Blom, K., Santoni, M. L., Lindauer, M., and Doerr, C. (2022b). PI is back! switching acquisition functions in bayesian optimization. In *Gaussian Processes, Spatiotemporal Modeling, and Decision-making Systems Workshop @ NeurIPS'22*.

Biedenkapp, A., Bozkurt, H. F., Eimer, T., Hutter, F., and Lindauer, M. (2020). Dynamic algorithm configuration: Foundation of a new meta-algorithmic framework. In *Proc. of ECAI'20*, pages 427–434.

Bossek, J., Doerr, C., and Kerschke, P. (2020). Initial design strategies and their effects on sequential model-based optimization: An exploratory case study based on BBOB. In *Proc. of GECCO'20*, pages 778–786.

Brochu, E., Cora, V., and de Freitas, N. (2010). A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv:1012.2599v1 [cs.LG]*.

Casalicchio, G., Bossek, J., Lang, M., Kirchhoff, D., Kerschke, P., Hofner, B., Seibold, H., Vanschoren, J., and Bischl, B. (2017). OpenML: An R package to connect to the machine learning platform OpenML. *Computational Statistics*, 32(3).

Cowen-Rivers, A., Lyu, W., Tutunov, R., Wang, Z., Grosnit, A., Griffiths, R., Maraval, A., Jianye, H., Wang, J., Peters, J., and Ammar, H. (2021). An empirical study of assumptions in Bayesian optimisation. *arXiv:2012.03826 [cs.LG]*.

Doerr, B. and Doerr, C. (2020). Theory of parameter control for discrete black-box optimization: Provable performance gains through dynamic parameter choices. In Doerr, B. and Neumann, F., editors, *Theory of Evolutionary Computation*, pages 271–321. Springer.

Eggensperger, K., Müller, P., Mallik, N., Feurer, M., Sass, R., Klein, A., Awad, N., Lindauer, M., and Hutter, F. (2021). HPOBench: A collection of reproducible multi-fidelity benchmark problems for HPO. In *Proc. of NeurIPS'21 Datasets and Benchmarks Track*.

Feurer, M., van Rijn, J., Kadra, A., Gijsbers, P., Mallik, N., Ravi, S., Müller, A., Vanschoren, J., and Hutter, F. (2021). OpenML-Python: an extensible Python API for OpenML. *JMLR*, 22(100):1–5.

Finck, S., Hansen, N., Ros, R., and Auger, A. (2009). Real-parameter black-box optimization benchmarking 2010: Presentation of the noiseless functions. `https://numbbo.github.io/gforge/downloads/download16.00/bbobdocfunctions.pdf`.

Forrester, A. I. J., Sóbester, A., and Keane, A. J. (2008). *Engineering Design via Surrogate Modelling - A Practical Guide*. John Wiley & Sons Ltd.

Frazier, P., Powell, W., and Dayanik, S. (2009). The knowledge-gradient policy for correlated normal beliefs. *Journal on Computing*, 21(4):599–613.

Garnett, R. (2023). *Bayesian Optimization.* Cambridge University Press.

Gutmann, H. (2001). A radial basis function method for global optimization. *J. Global Optimiz.*, 19(3):201–227.

Hansen, N., Auger, A., Ros, R., Mersman, O., Tušar, T., and Brockhoff, D. (2020). COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software.*

Hansen, N., Müller, S. D., and Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computing*, 11(1):1–18.

Hennig, P. and Schuler, C. (2012). Entropy search for information-efficient global optimization. *JMLR*, 98888(1):1809–1837.

Hoffman, M., Brochu, E., and de Freitas, N. (2014). Portfolio allocation for Bayesian optimization. In *Proc. of UAI'11*, pages 327–336.

Hutter, F., Hoos, H., and Leyton-Brown, K. (2014). An efficient approach for assessing hyperparameter importance. In *Proc. of ICML'14*, pages 754–762.

Kandasamy, K., Vysyaraju, K., Neiswanger, W., Paria, B., Collins, C., Schneider, J., Poczos, B., and Xing, E. (2020). Tuning hyperparameters without grad students: Scalable and robust Bayesian optimisation with dragonfly. *Journal of Machine Learning Research*, 21:81:1–81:27.

Karafotias, G., Hoogendoorn, M., and Eiben, Á. (2015). Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Trans. Evolutionary Computation*, 19(2):167–187.

Kushner, H. (1964). A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Fluids Engineering*, pages 97–106.

Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Ruhkopf, T., Sass, R., and Hutter, F. (2022). SMAC3: A versatile bayesian optimization package for Hyperparameter Optimization. *JMLR*, 23(54):1–9.

Lindauer, M., Feurer, M., Eggensperger, K., Biedenkapp, A., and Hutter, F. (2019). Towards assessing the impact of bayesian optimization's own hyperparameters. In *IJCAI'19 DSO Workshop*.

Liu, X., Wan, D., and Chen, G. (2018). Kriging-Based surrogate model combined with weighted expected improvement for ship hull form optimization. *ASME 2018 37th International Conference on Ocean, Offshore and Arctic Engineering*.

Makarova, A., Shen, H., Perrone, V., Klein, A., Faddoul, J., Krause, A., Seeger, M., and Archambeau, C. (2022). Automatic termination for hyperparameter optimization. In Guyon, I., Lindauer, M., van der Schaar, M., Hutter, F., and Garnett, R., editors, *International Conference on Automated Machine Learning, AutoML 2022, 25-27 July 2022, Johns Hopkins University, Baltimore, MD, USA*, volume 188 of *Proceedings of Machine Learning Research*, pages 7/1–21. PMLR.

Mersmann, O., Bischl, B., Trautmann, H., Preuss, M., Weihs, C., and Rudolph, G. (2011). Exploratory landscape analysis. In *Proc. of GECCO'11*, pages 829–836.

Mockus, J. (1989). *Bayesian approach to global optimization. Theory and Applications.* Kluwer Academic Publishers.

Mockus, J., Tiesis, V., and Zilinskas, A. (1978). The application of Bayesian methods for seeking the extremum. *Towards Global Optimization*, 2(117-129).

Pushak, Y. and Hoos, H. (2018). Algorithm configuration landscapes: - more benign than expected? In *Proc. of PPSN'18*, pages 271–283.

Qin, C., Klabjan, D., and Russo, D. (2017). Improving the expected improvement algorithm. In *Proc. of NeurIPS'17*, pages 5381–5391.

Sobester, A., Leary, S., and Keane, A. (2005). On the design of optimization strategies based on global response surface approximation models. *J. Glob. Optim.*, 33(1):31–59.

Speck, D., Biedenkapp, A., Hutter, F., Mattmüller, R., and Lindauer, M. (2021). Learning heuristic selection with dynamic algorithm configuration. In *Proc. of ICAPS'21*.

Thompson, W. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294.

Volpp, M., Fröhlich, L., Fischer, K., Doerr, A., Falkner, S., Hutter, F., and Daniel, C. (2020). Meta-learning acquisition functions for transfer learning in Bayesian optimization. In *Proc. of ICLR'20*.

Xiao, S., Rotaru, M., and Sykulski, J. (2012). Exploration versus exploitation using kriging surrogate modelling in electromagnetic design. *COMPEL - The international journal for computation and mathematics in electrical and electronic engineering*, 31(5):1541–1551.

Xiao, S., Rotaru, M., and Sykulski, J. (2013). Adaptive weighted expected improvement with rewards approach in kriging assisted electromagnetic design. *IEEE Transactions on Magnetics*, 49(5):2057–2060.

7 Submission Checklist

1. For all authors…

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]

   (b) Did you describe the limitations of your work? [Yes]

   (c) Did you discuss any potential negative societal impacts of your work? [Yes]

   (d) Have you read the ethics author's and review guidelines and ensured that your paper conforms to them? `https://automl.cc/ethics-accessibility/` [Yes]

2. If you are including theoretical results…

   (a) Did you state the full set of assumptions of all theoretical results? [N/A] No theoretical results.

   (b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments…

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit version), an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [Yes]

   (b) Did you include the raw results of running the given instructions on the given code and data? [No] We will upload the datasets upon acceptance.

(c) Did you include scripts and commands that can be used to generate the figures and tables in your paper based on the raw results of the code, data, and instructions given? [Yes]

(d) Did you ensure sufficient code quality such that your code can be safely executed and the code is properly documented? [Yes]

(e) Did you specify all the training details (e.g., data splits, pre-processing, search spaces, fixed hyperparameter settings, and how they were chosen)? [Yes]

(f) Did you ensure that you compared different methods (including your own) exactly on the same benchmarks, including the same datasets, search space, code for training and hyperparameters for that code? [Yes]

(g) Did you run ablation studies to assess the impact of different components of your approach? [Yes] See main.

(h) Did you use the same evaluation protocol for the methods being compared? [Yes]

(i) Did you compare performance over time? [Yes] Rank over time

(j) Did you perform multiple runs of your experiments and report random seeds? [Yes] 10 seeds

(k) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

(l) Did you use tabular or surrogate benchmarks for in-depth evaluations? [Yes]

(m) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

(n) Did you report how you tuned hyperparameters, and what time and resources this required (if they were not automatically tuned by your AutoML method, e.g. in a NAS approach; and also hyperparameters of your own method)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets…

(a) If your work uses existing assets, did you cite the creators? [N/A]

(b) Did you mention the license of the assets? [N/A]

(c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects…

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A How To Detect the Adjustment Point

We determine the point to adjust based on the Upper Bound Regret (UBR) (Makarova et al., 2022). First, because the signal is rugged, we smooth it by applying a moving interquartile mean (25 %-75 % quartiles) with window size 7 to UBR. Then, we calculate the absolute gradients. We signal time to adjust when the last absolute gradient is close to 0 with an absolute tolerance of ϵ times the last observed maximum of the absolute gradient.

B Hardware and Runtime

All experiments are conducted on a CPU cluster with 990 nodes with AMD Milan 7763 CPUs. The compute time for the BBOB 8d functions was 45 min each so 14 040 h = 585 d in total on CPU (including ablation). The compute time for the HPOBench was 90 sec each so 288 h = 12 d in total on CPU.

C BBOB Results

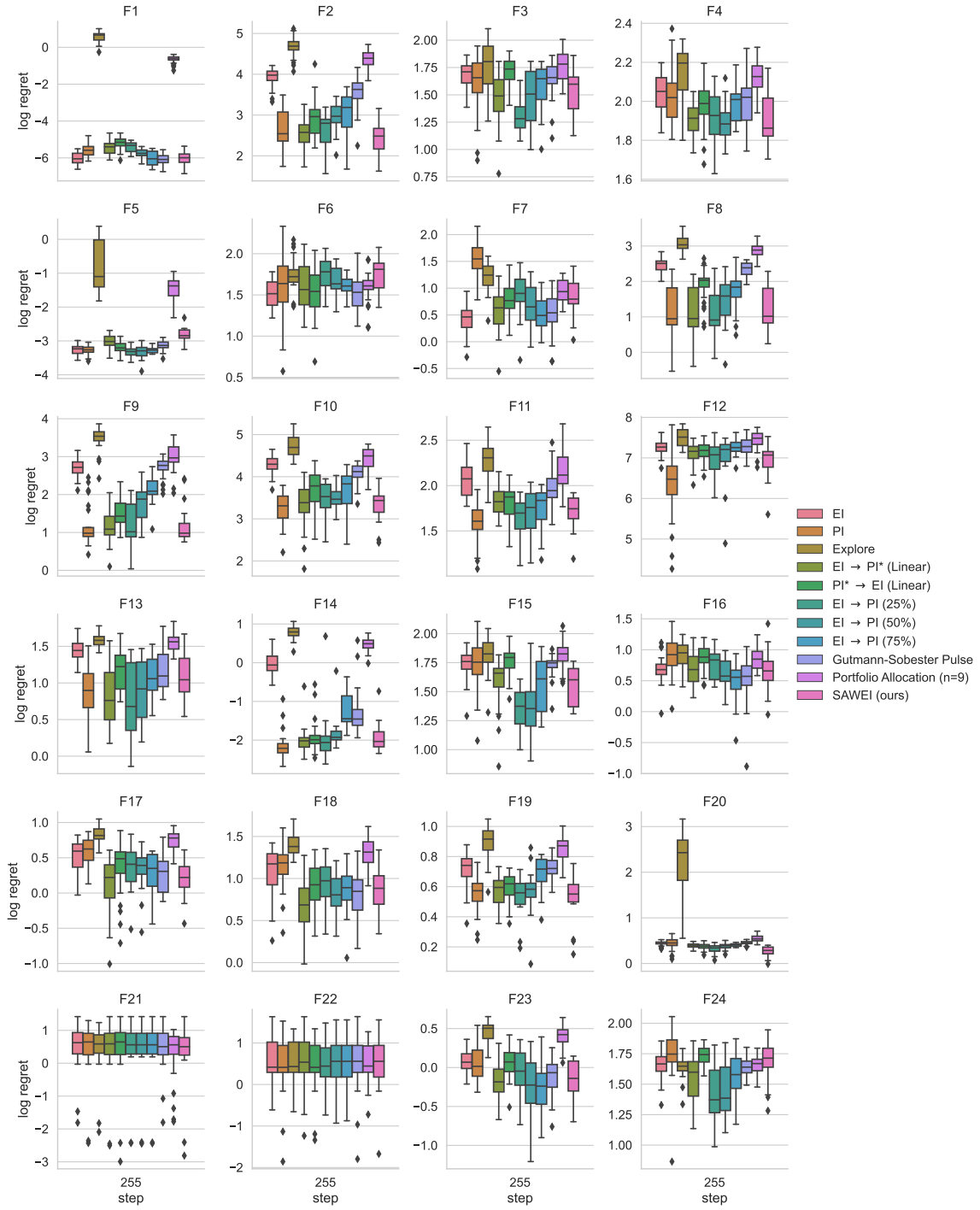


Figure 7: Final log regret on BBOB (8d, 10 seeds, 3 instances)

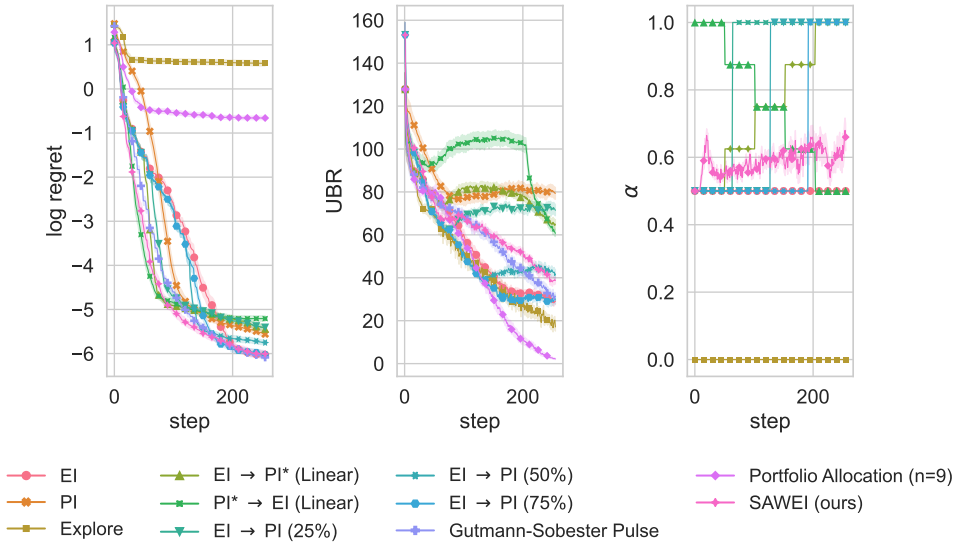


Figure 8: BBOB Function 1

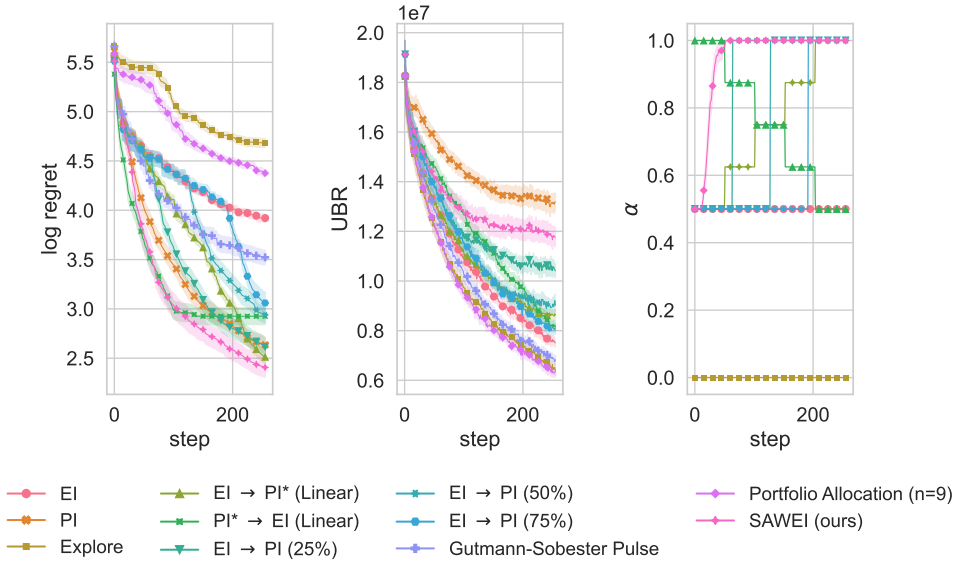


Figure 9: BBOB Function 2

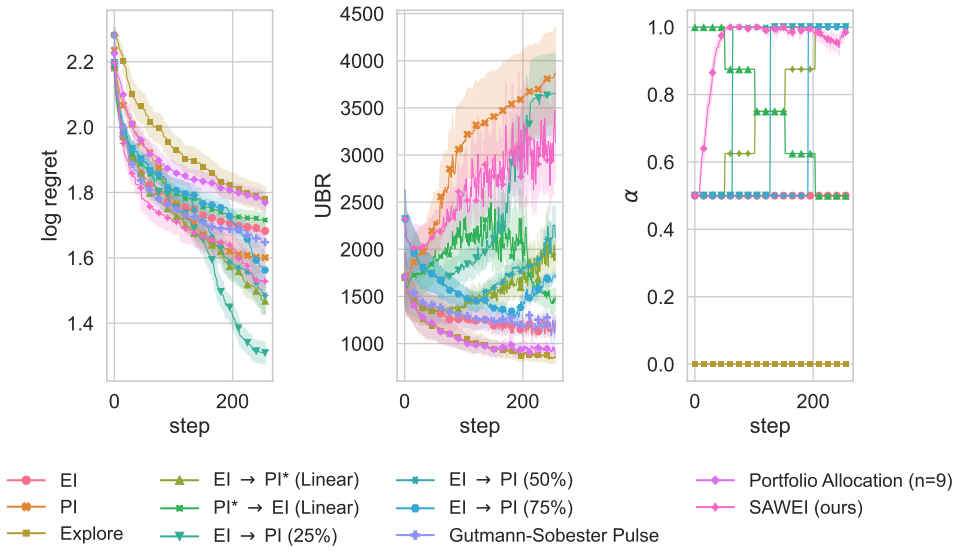


Figure 10: BBOB Function 3

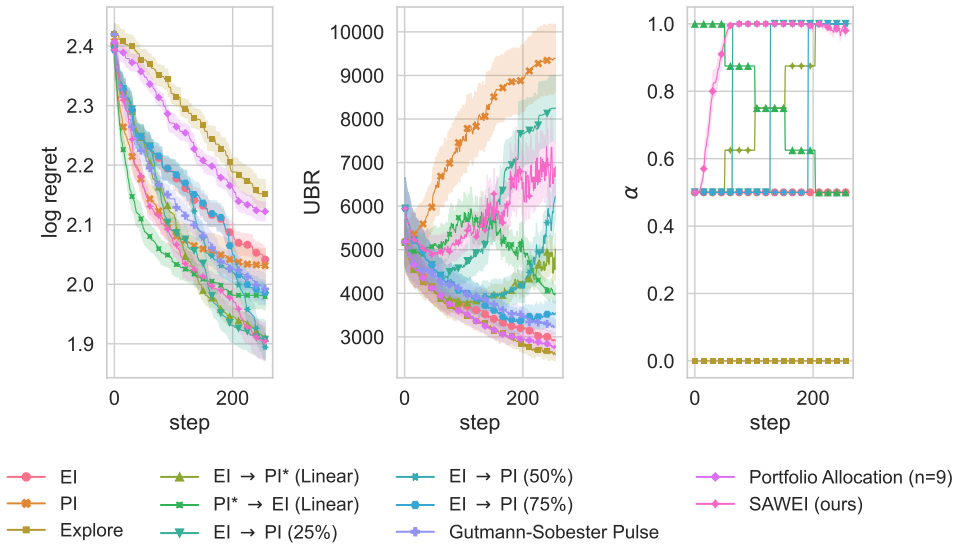


Figure 11: BBOB Function 4

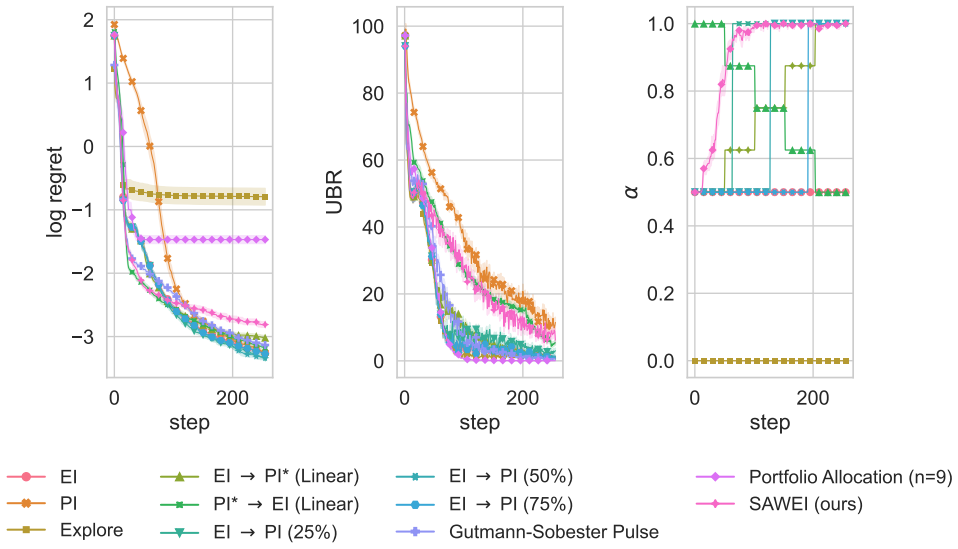


Figure 12: BBOB Function 5

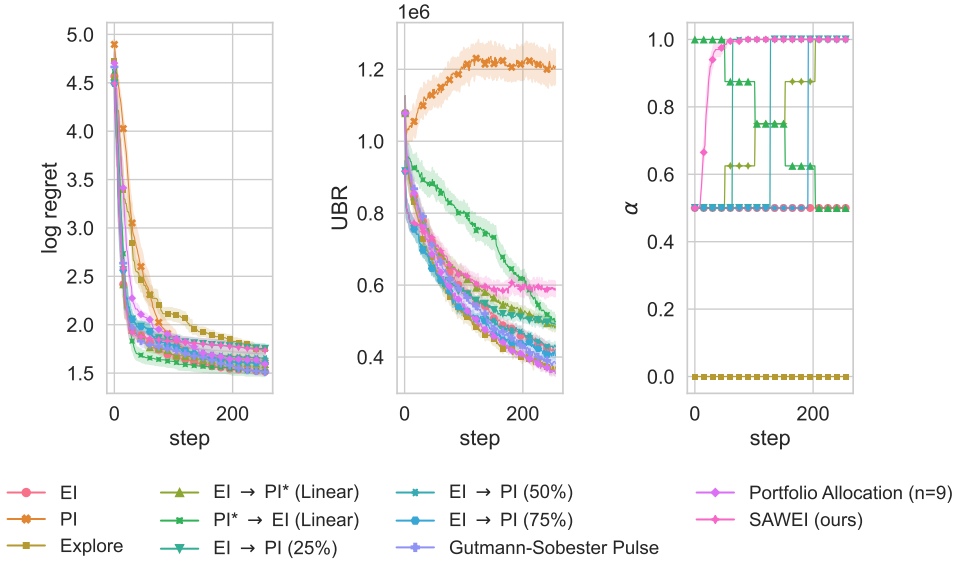


Figure 13: BBOB Function 6

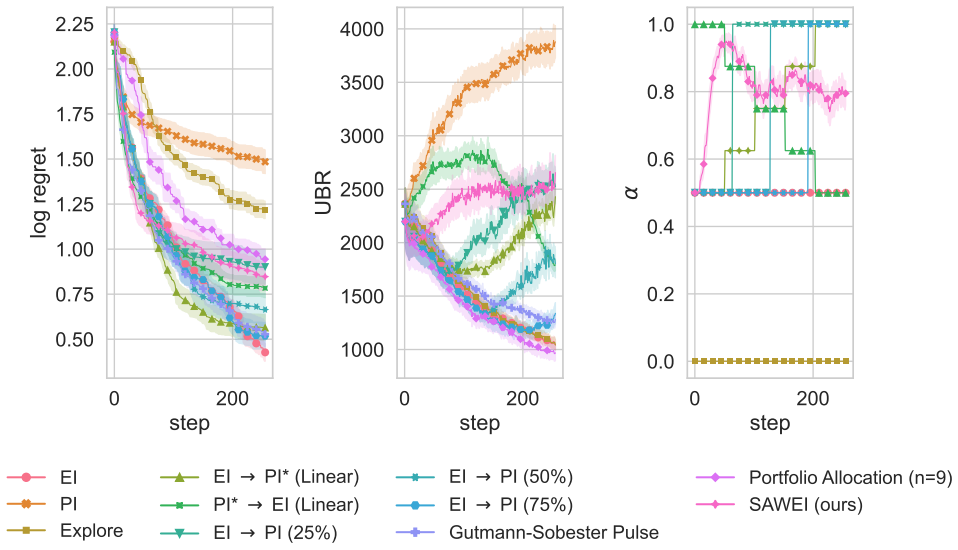


Figure 14: BBOB Function 7

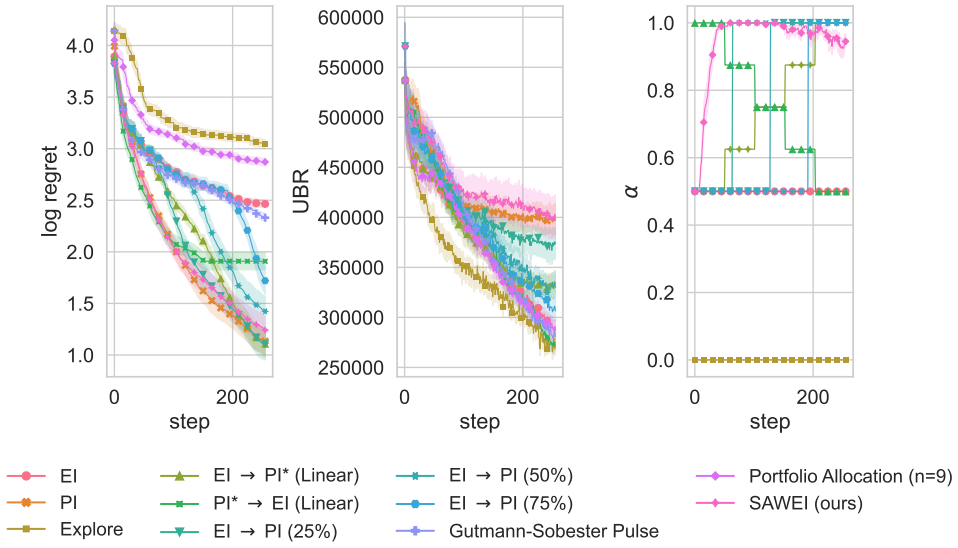


Figure 15: BBOB Function 8

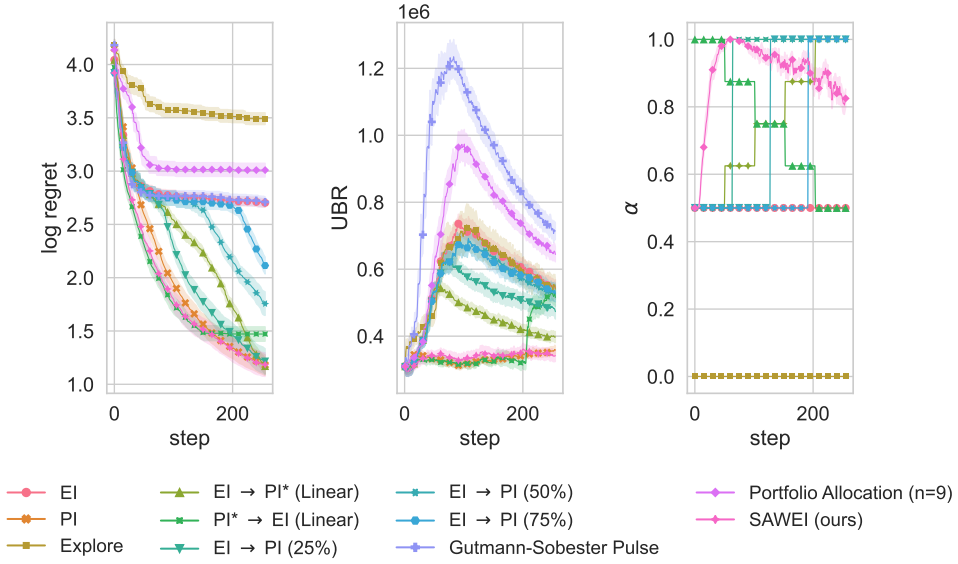


Figure 16: BBOB Function 9

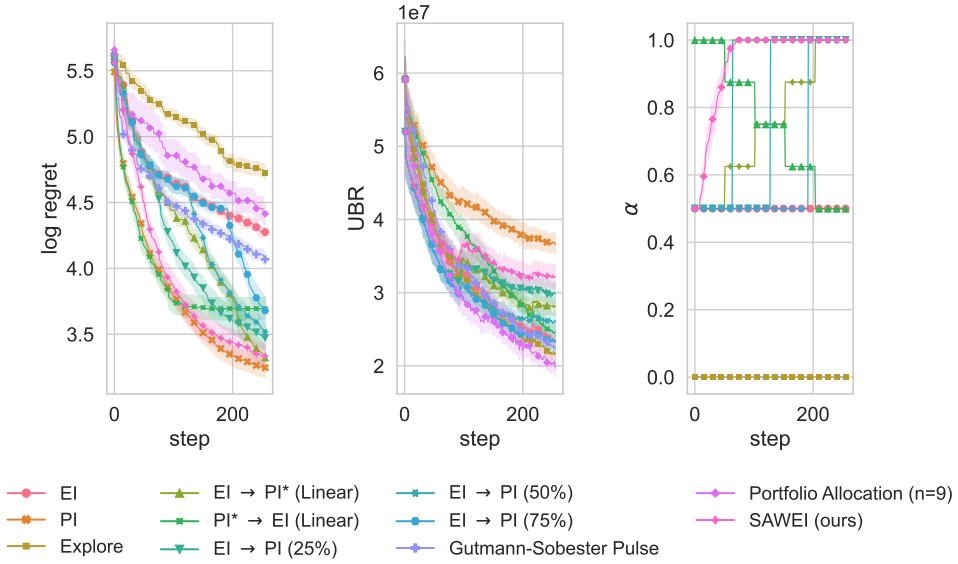


Figure 17: BBOB Function 10

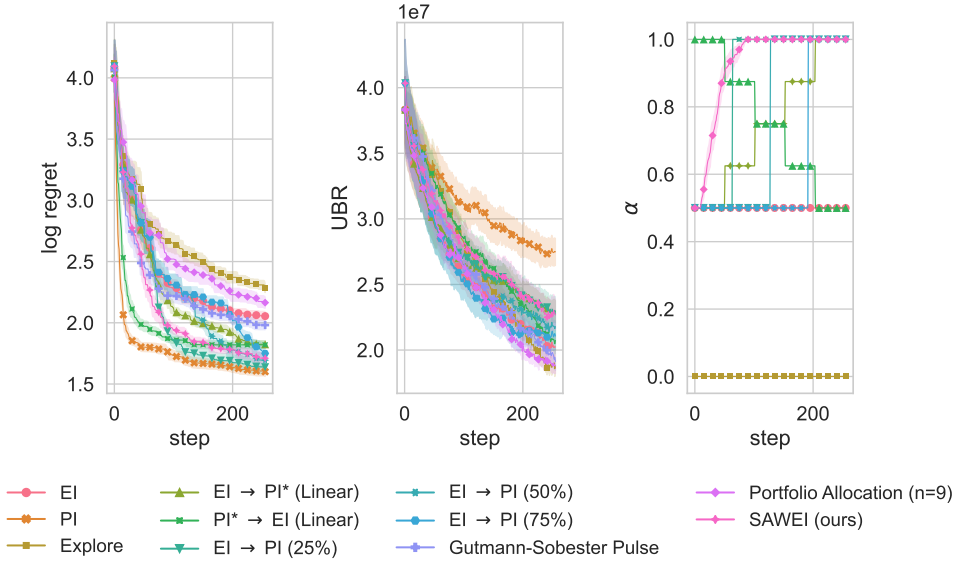


Figure 18: BBOB Function 11

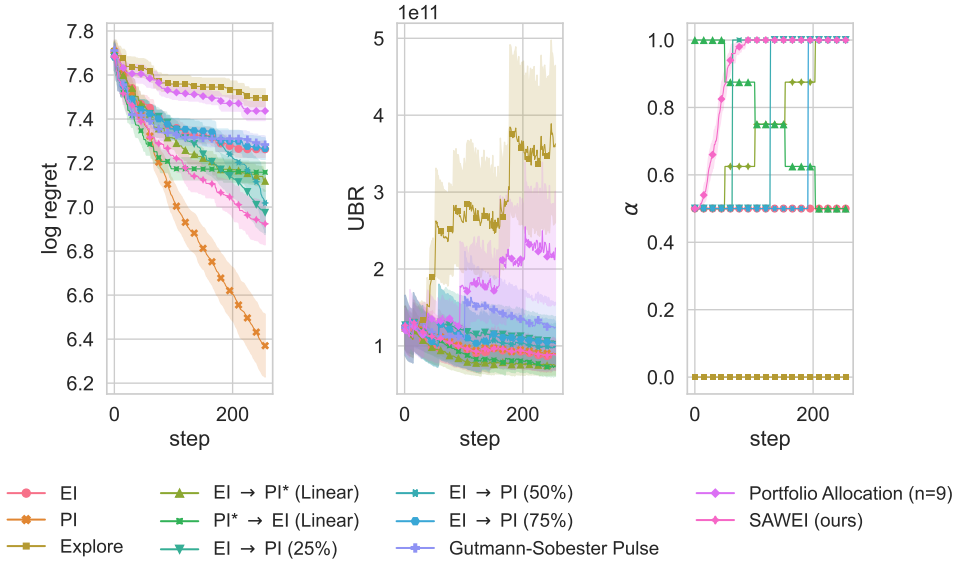


Figure 19: BBOB Function 12

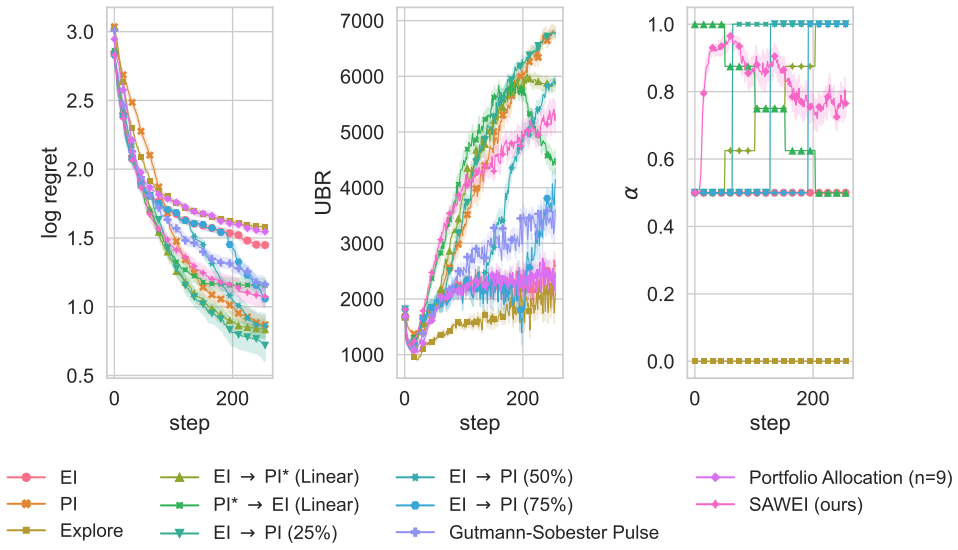


Figure 20: BBOB Function 13

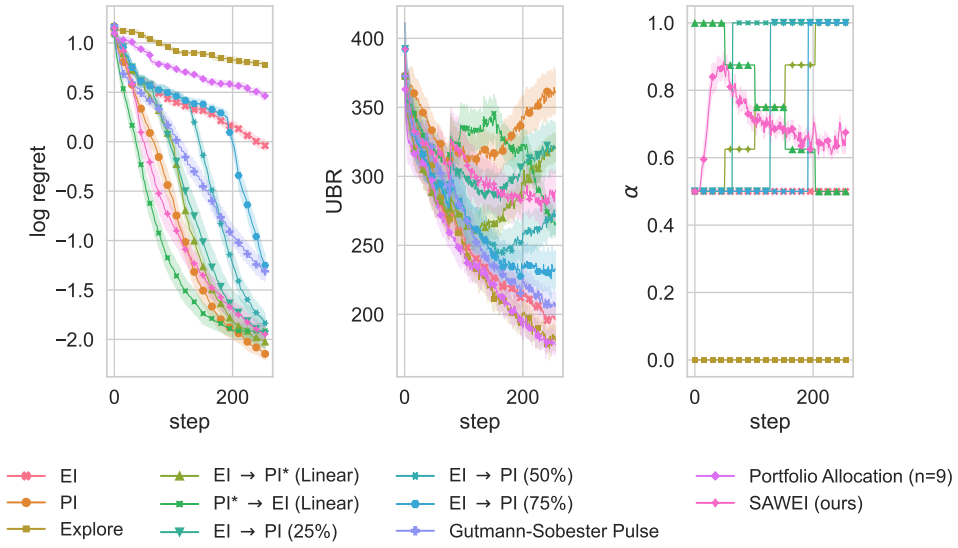


Figure 21: BBOB Function 14

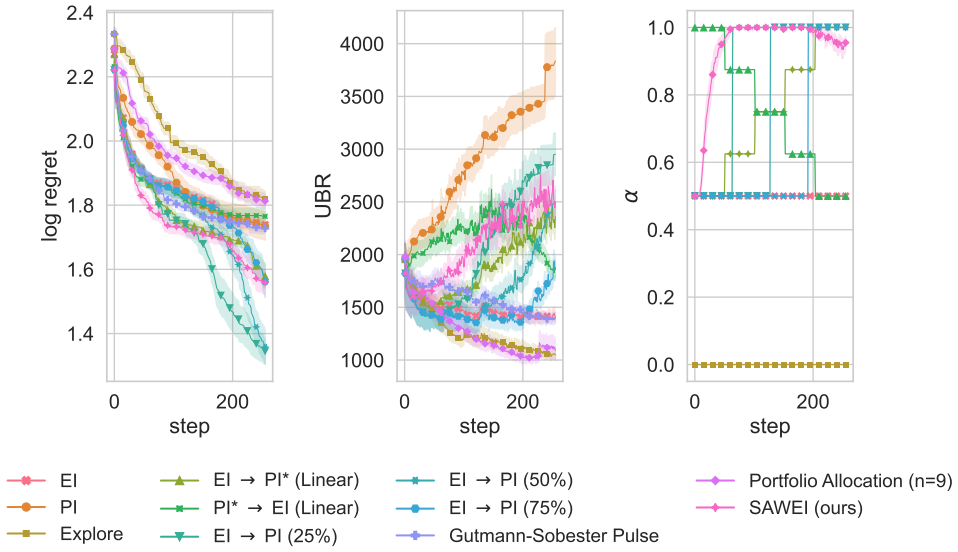


Figure 22: BBOB Function 15

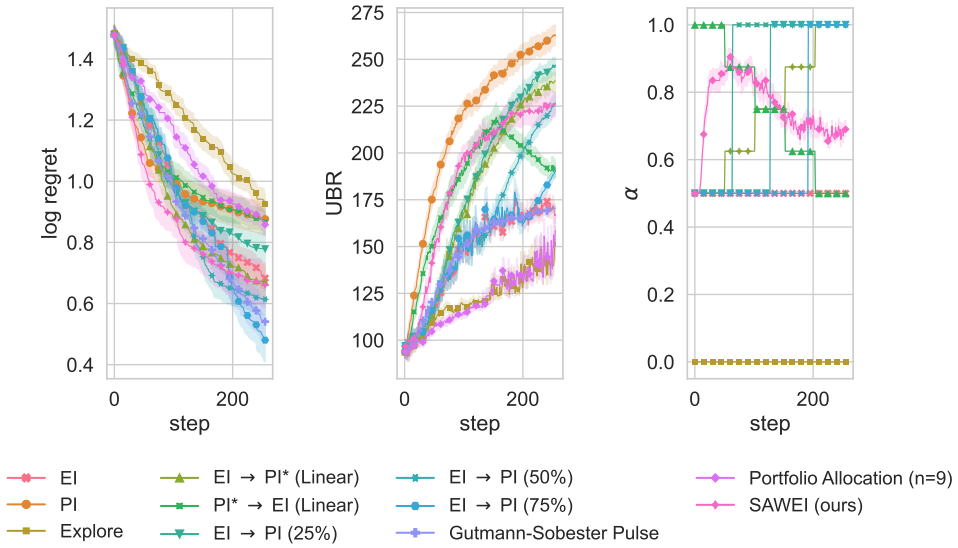


Figure 23: BBOB Function 16

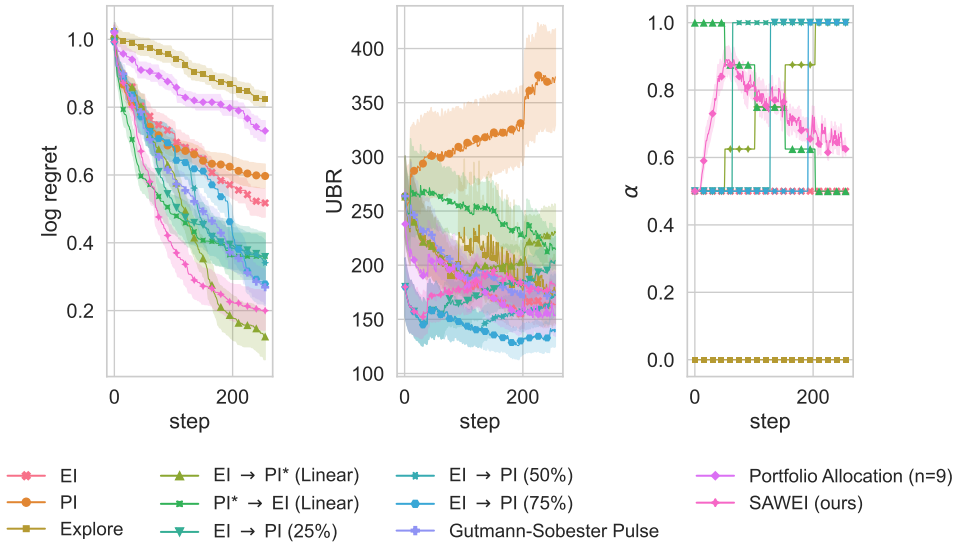


Figure 24: BBOB Function 17

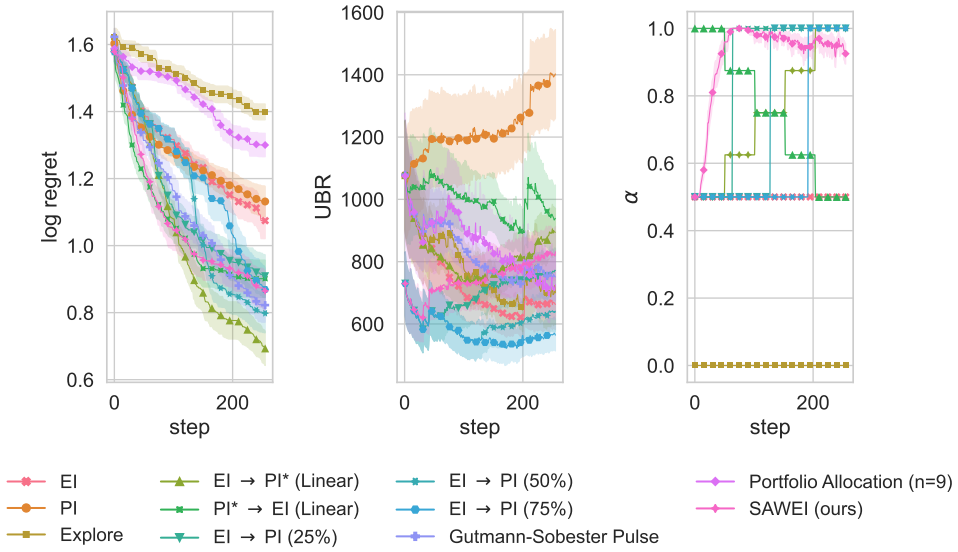


Figure 25: BBOB Function 18

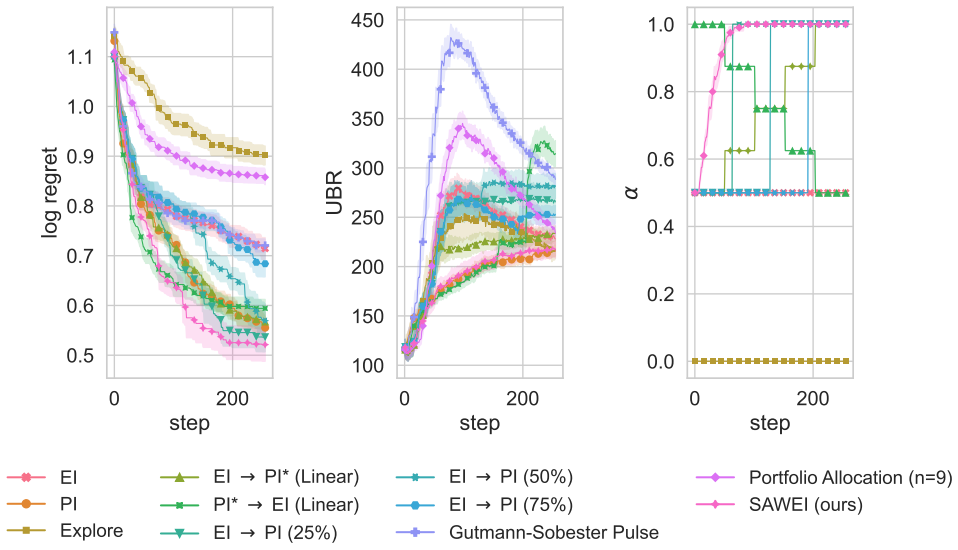


Figure 26: BBOB Function 19

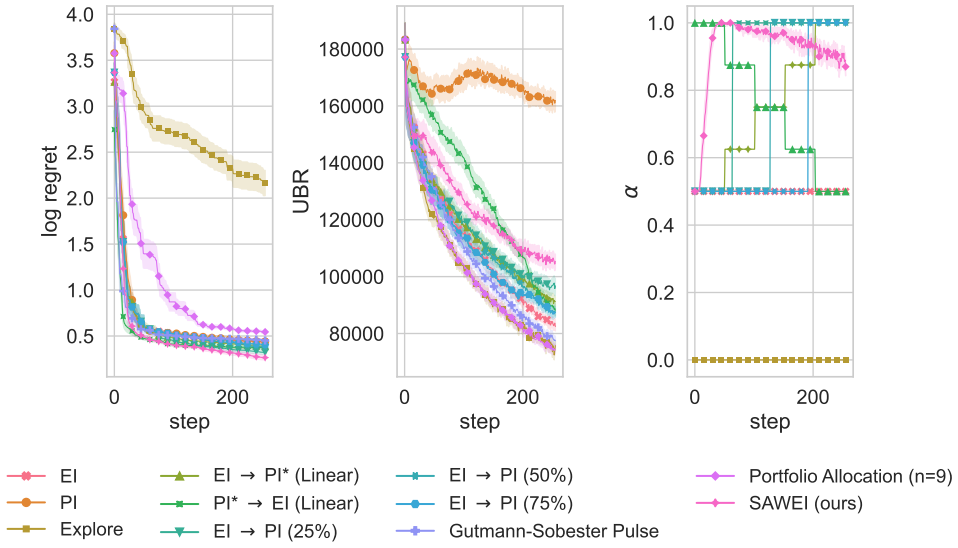


Figure 27: BBOB Function 20

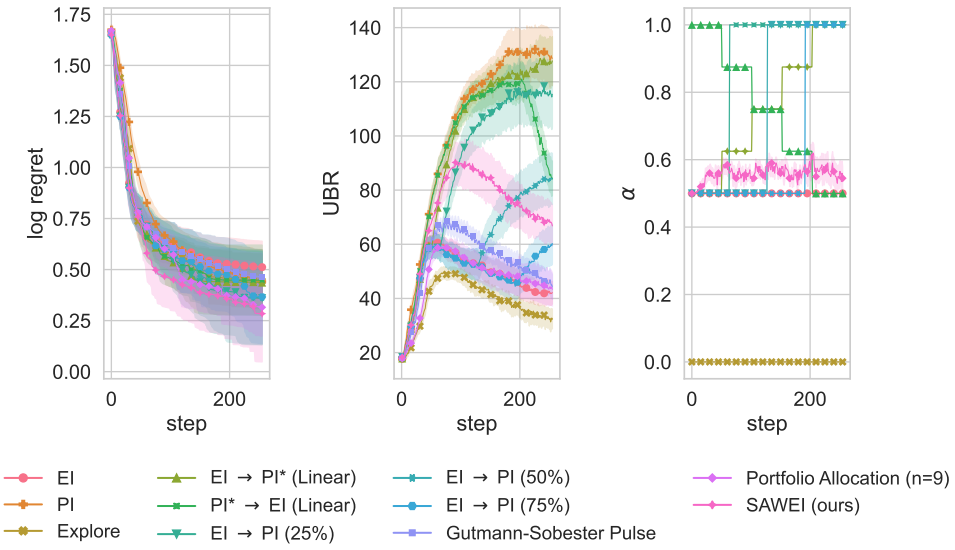


Figure 28: BBOB Function 21

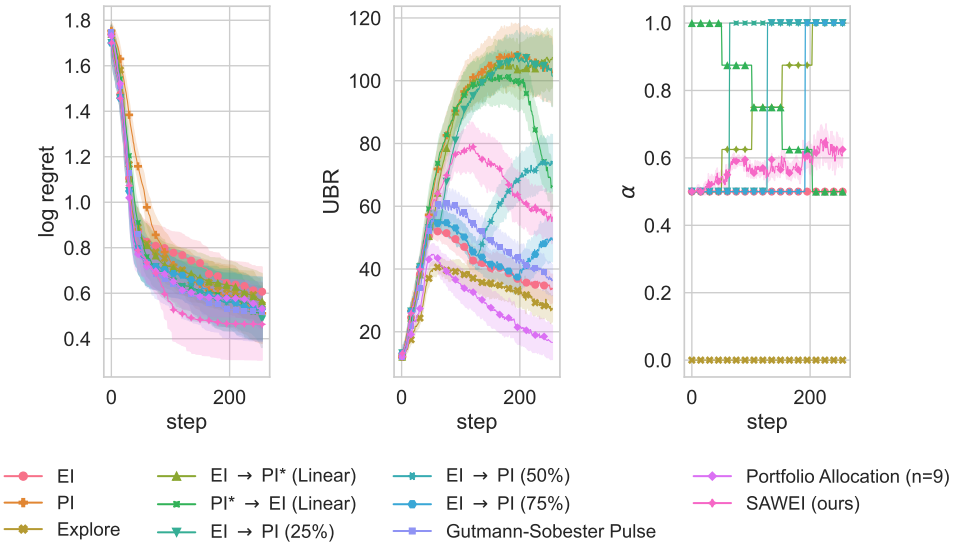


Figure 29: BBOB Function 22

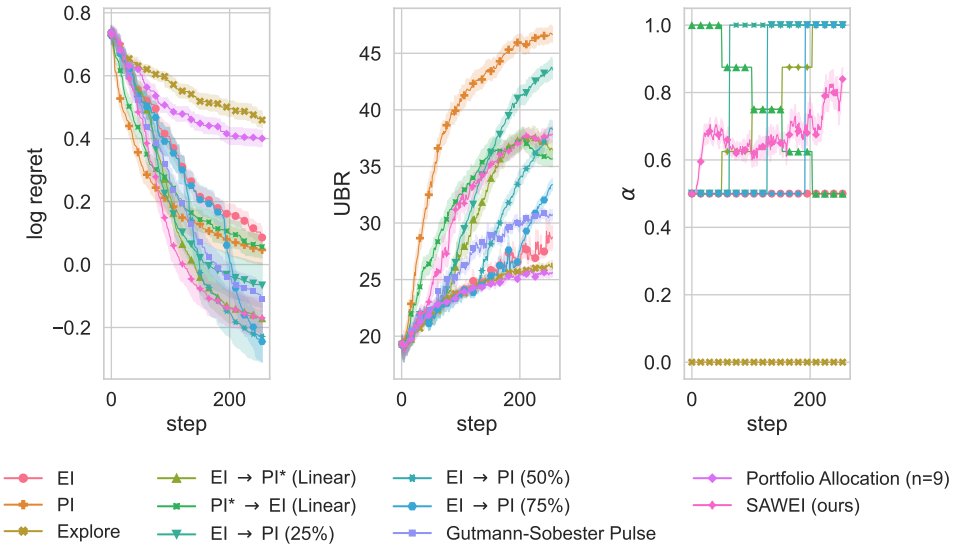


Figure 30: BBOB Function 23

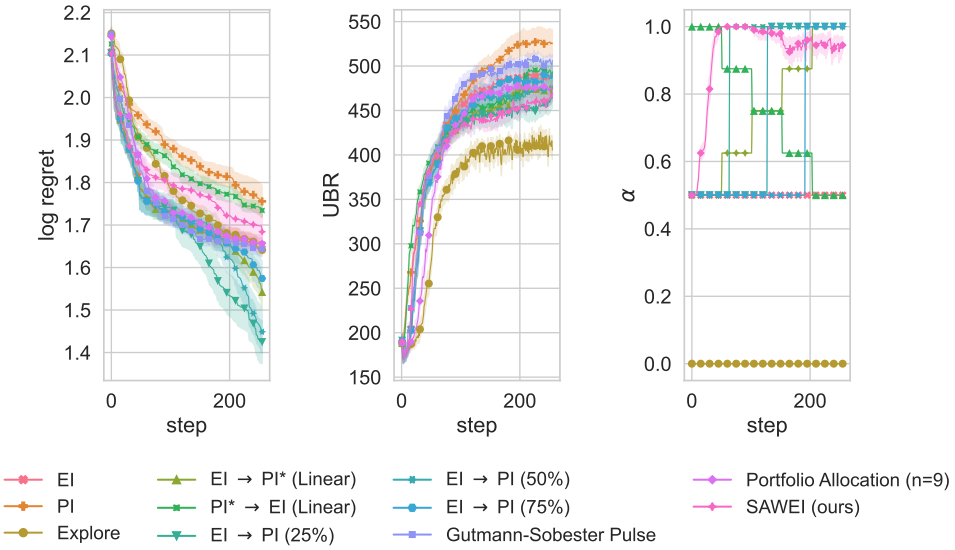


Figure 31: BBOB Function 24

D HPOBench Results

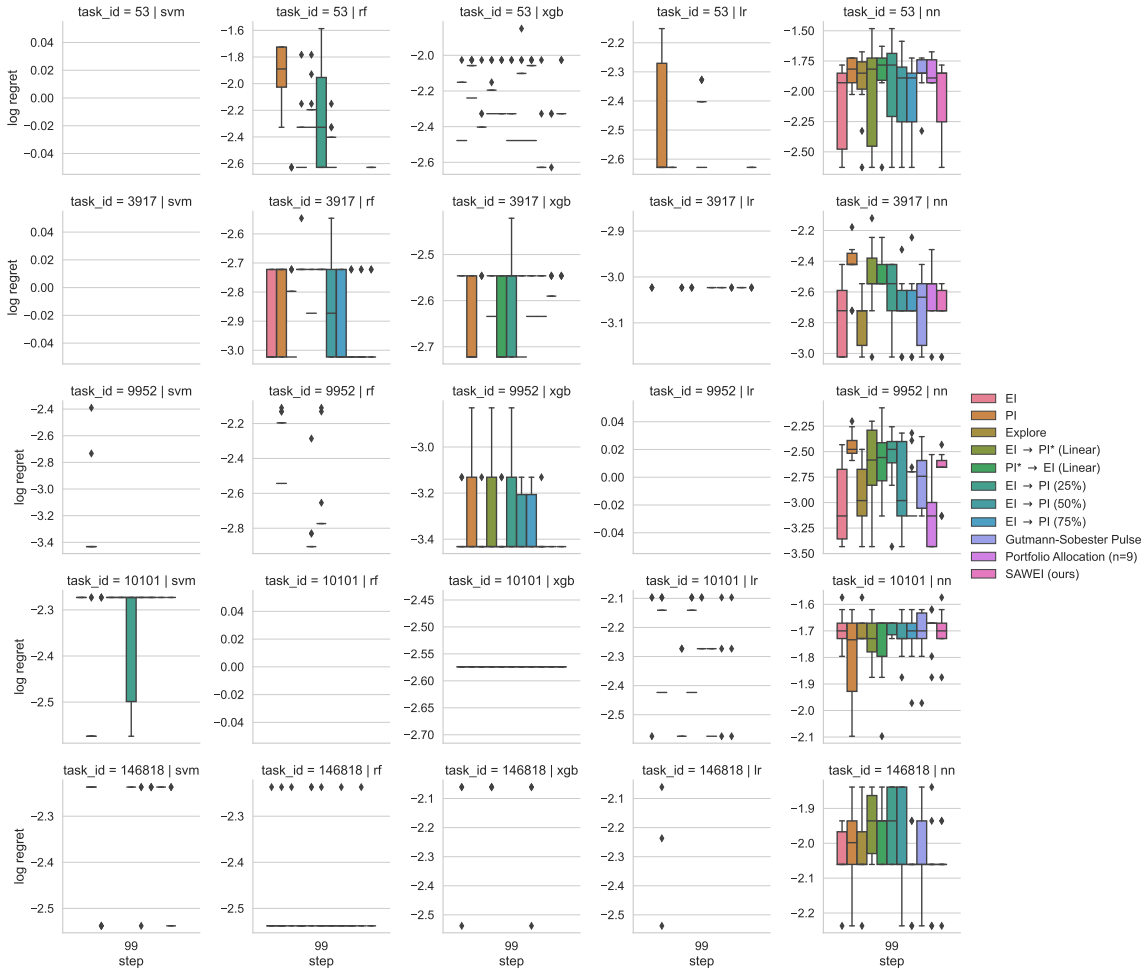


Figure 32: Final log regret on HPOBench (10 seeds). Please note that on this tabular benchmark a log regret of 0 can be achieved which is not plotted.

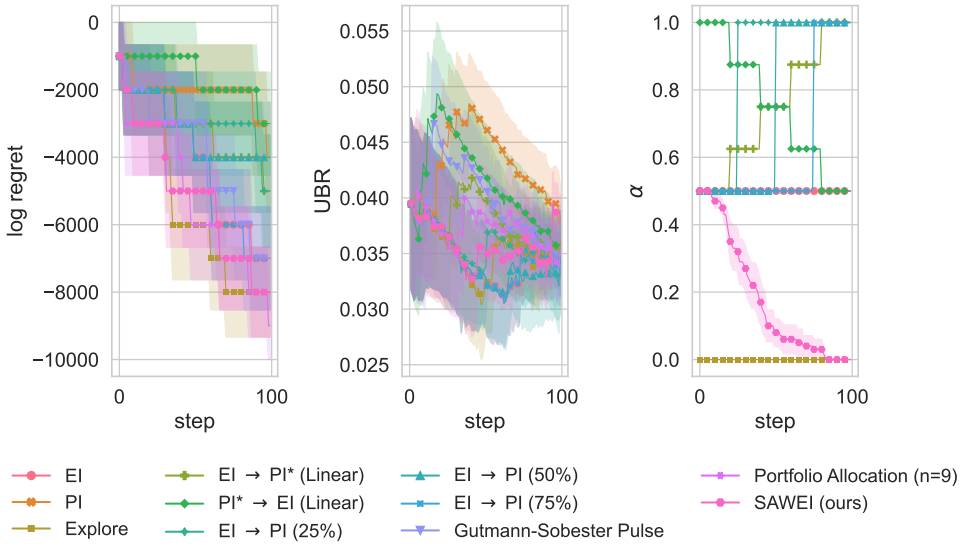


Figure 33: HPOBench ML: (model, task_id) = ('lr', 10101)

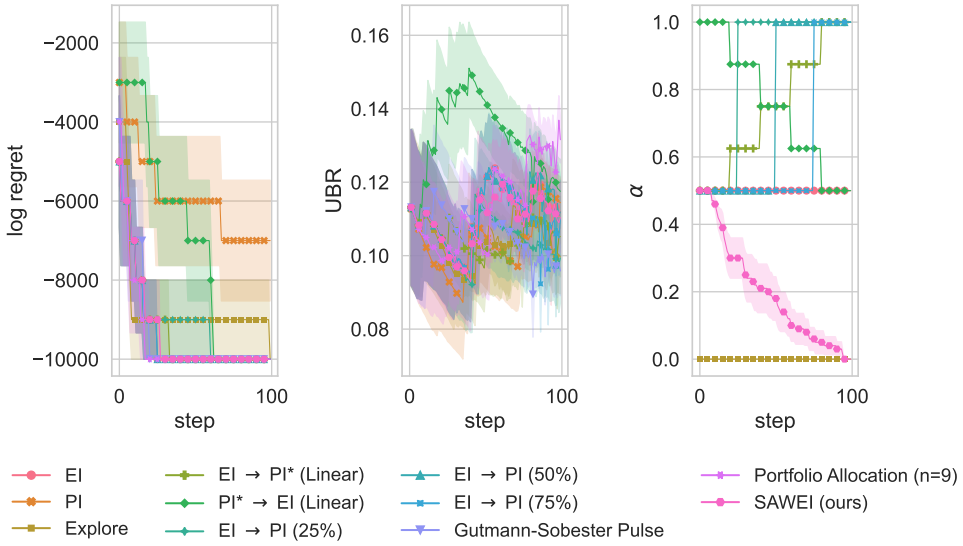


Figure 34: HPOBench ML: (model, task_id) = ('lr', 146818)

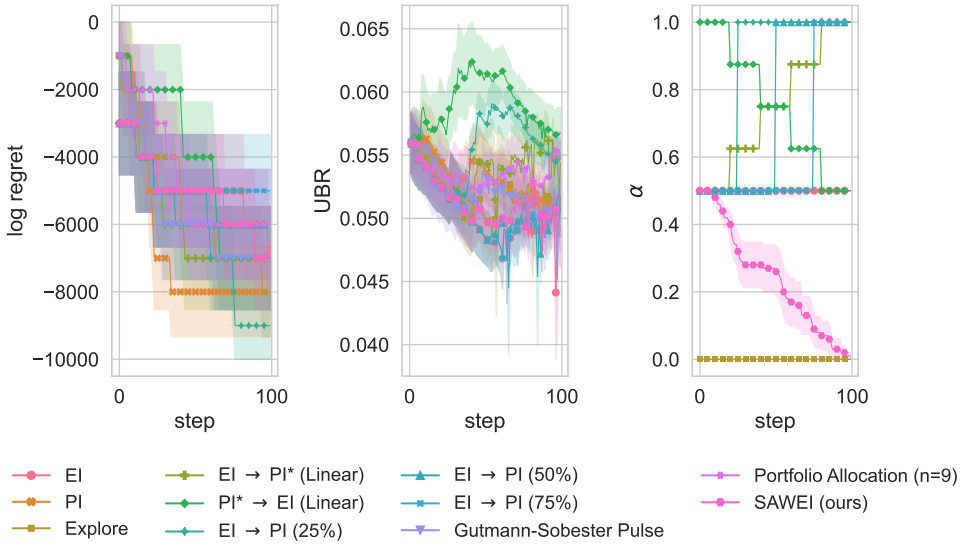


Figure 35: HPOBench ML: (model, task_id) = ('lr', 3917)

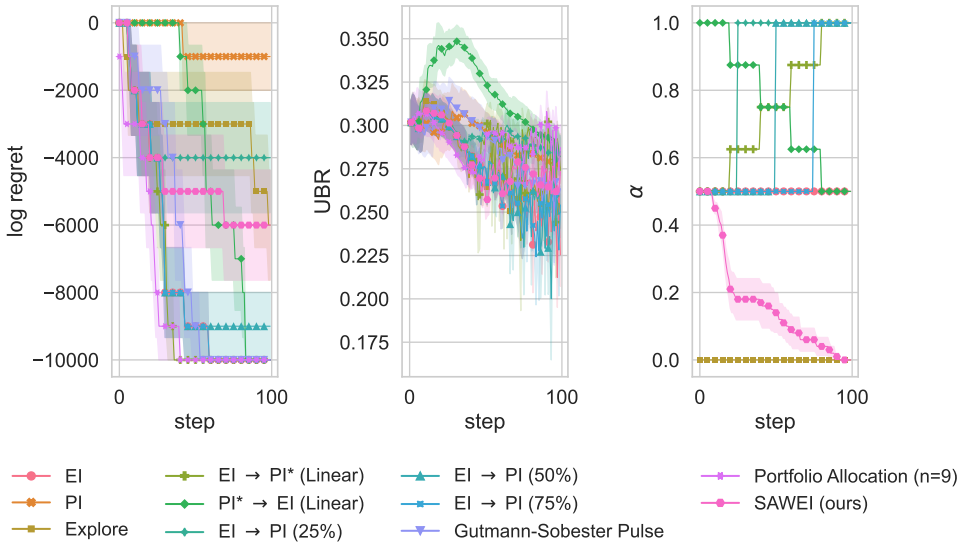


Figure 36: HPOBench ML: (model, task_id) = ('lr', 53)

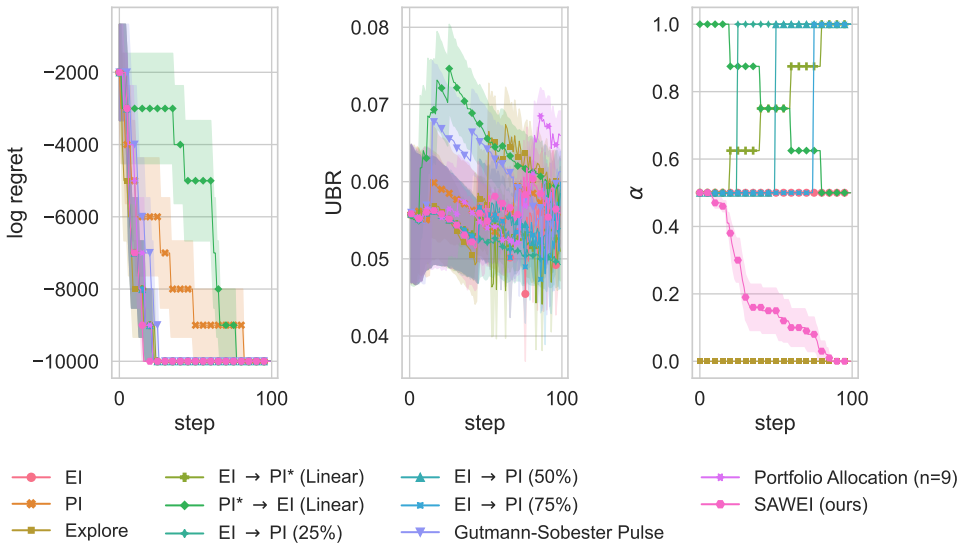


Figure 37: HPOBench ML: (model, task_id) = ('lr', 9952)

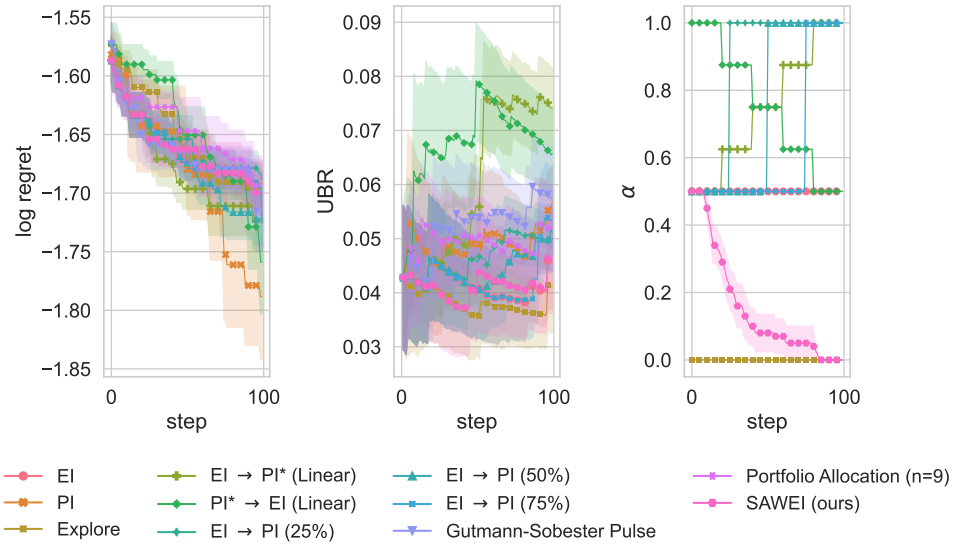


Figure 38: HPOBench ML: (model, task_id) = ('nn', 10101)

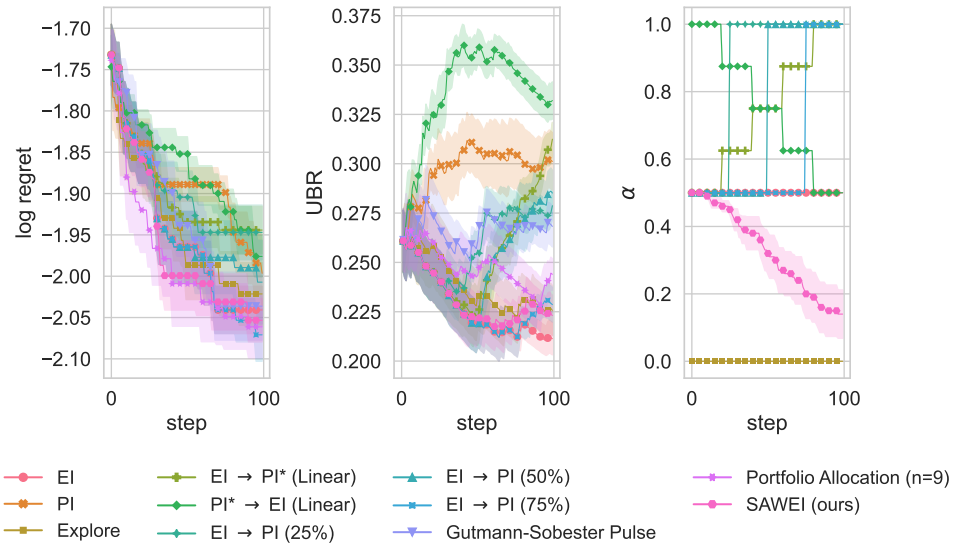


Figure 39: HPOBench ML: (model, task_id) = ('nn', 146818)

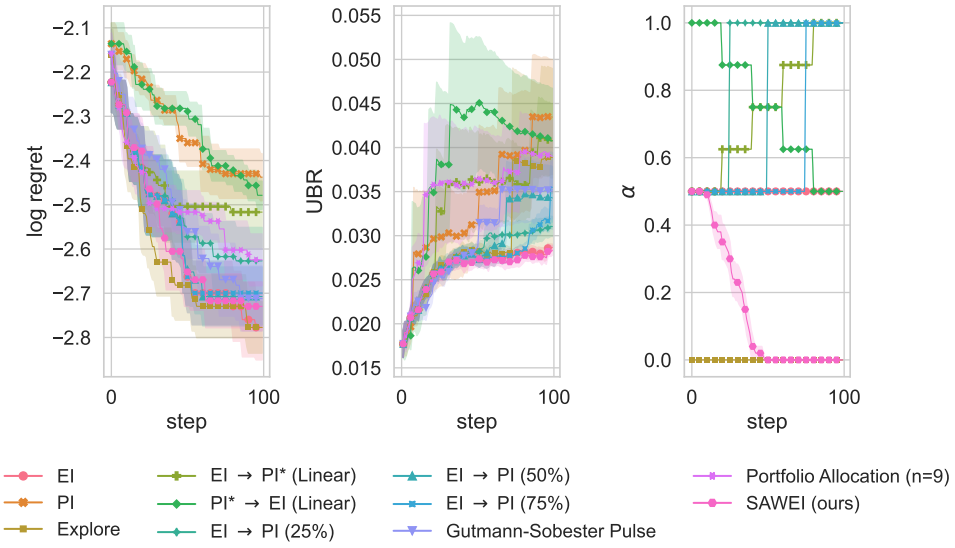


Figure 40: HPOBench ML: (model, task_id) = ('nn', 3917)

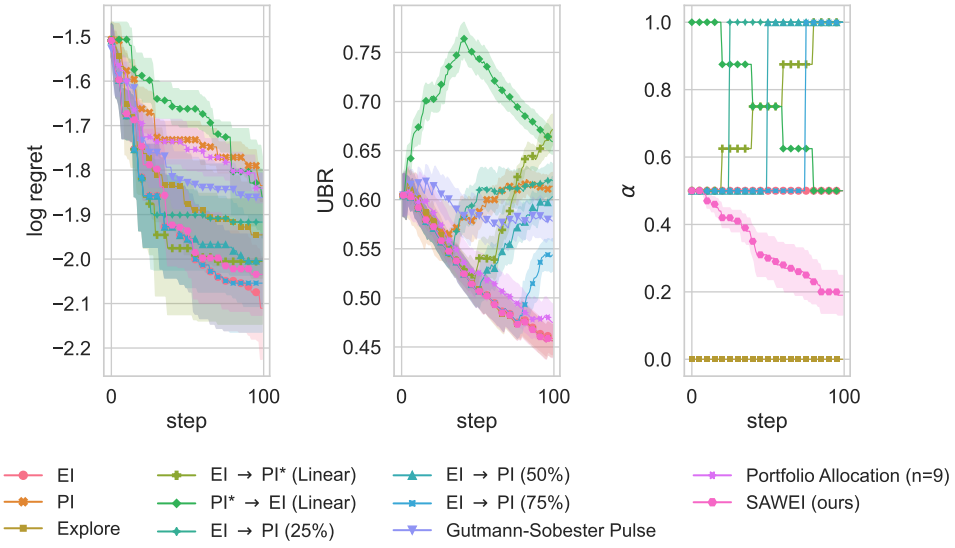


Figure 41: HPOBench ML: (model, task_id) = ('nn', 53)

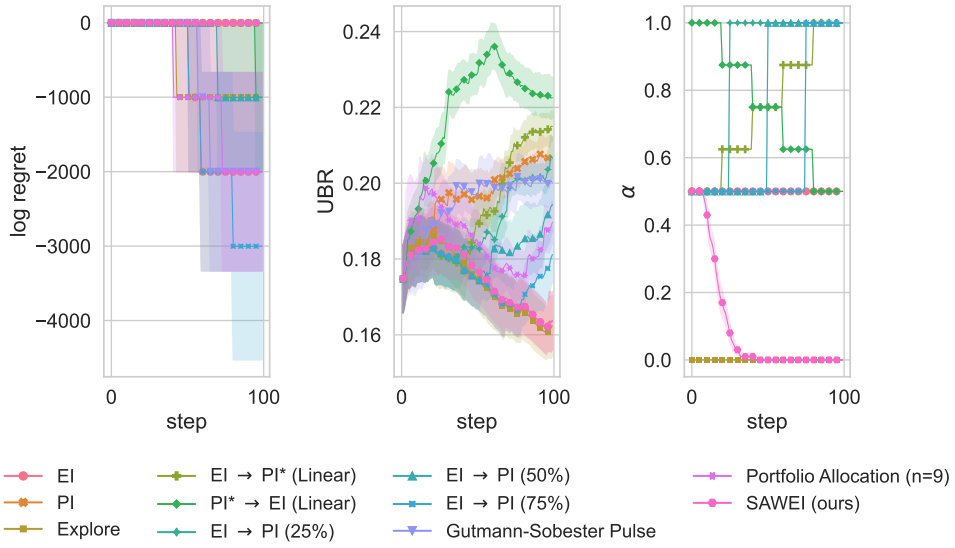


Figure 42: HPOBench ML: (model, task_id) = ('nn', 9952)

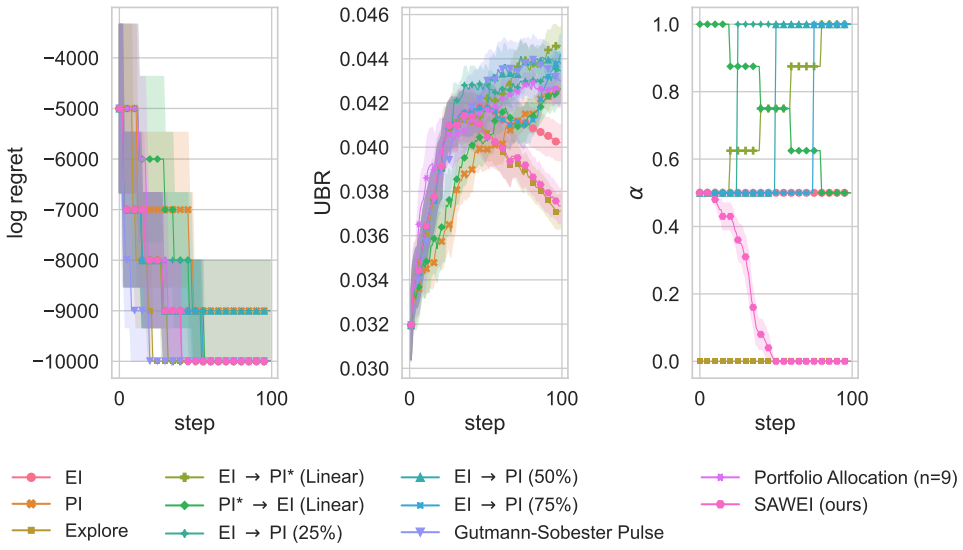


Figure 43: HPOBench ML: (model, task_id) = ('rf', 10101)

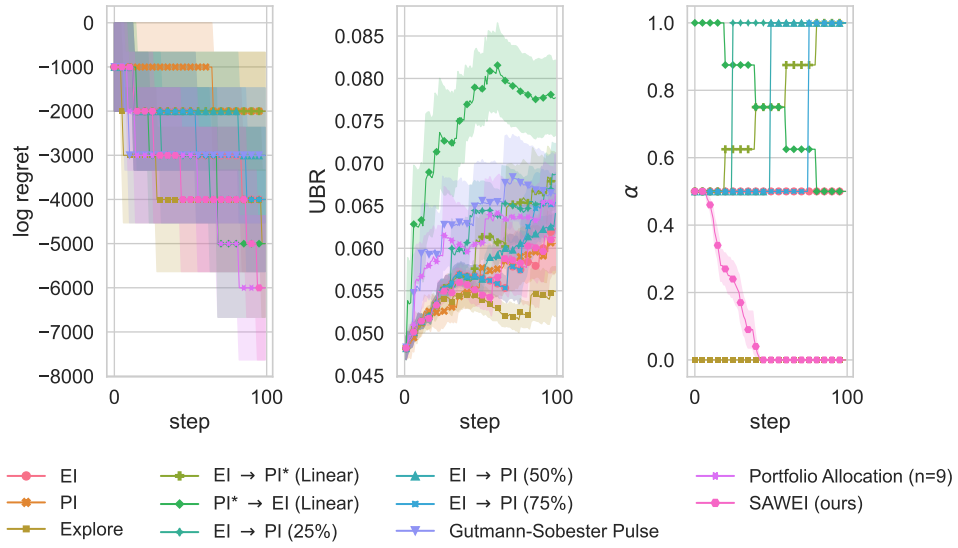


Figure 44: HPOBench ML: (model, task_id) = ('rf', 146818)

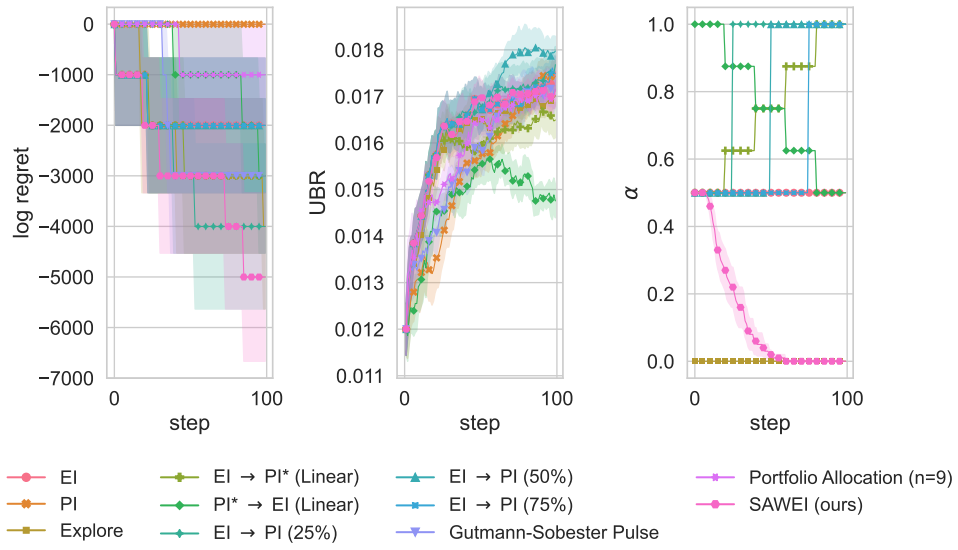


Figure 45: HPOBench ML: (model, task_id) = ('rf', 3917)

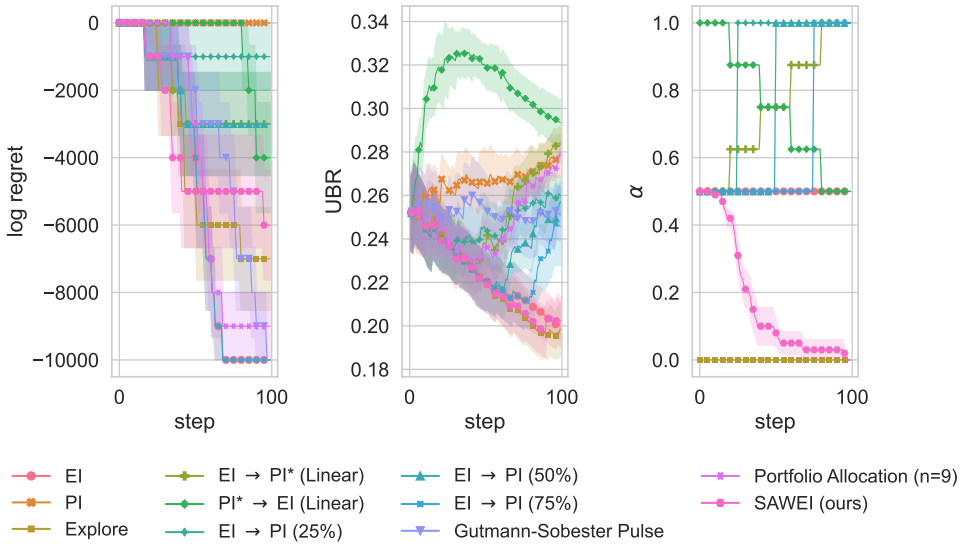


Figure 46: HPOBench ML: (model, task_id) = ('rf', 53)

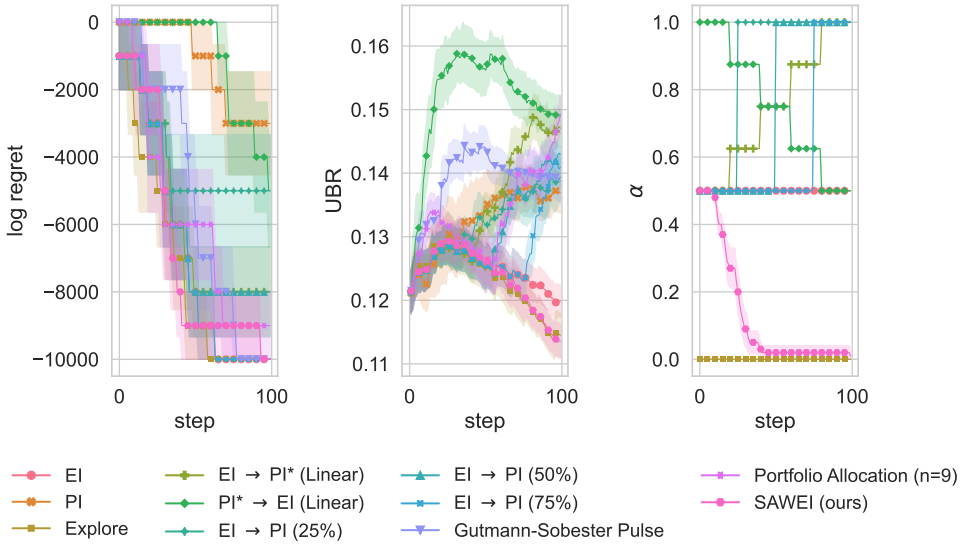


Figure 47: HPOBench ML: (model, task_id) = ('rf', 9952)

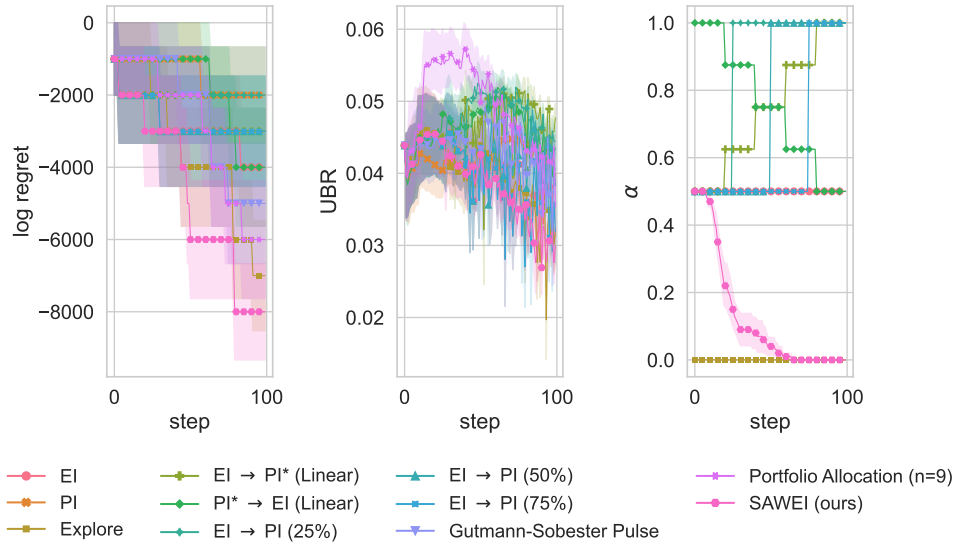


Figure 48: HPOBench ML: (model, task_id) = ('svm', 10101)

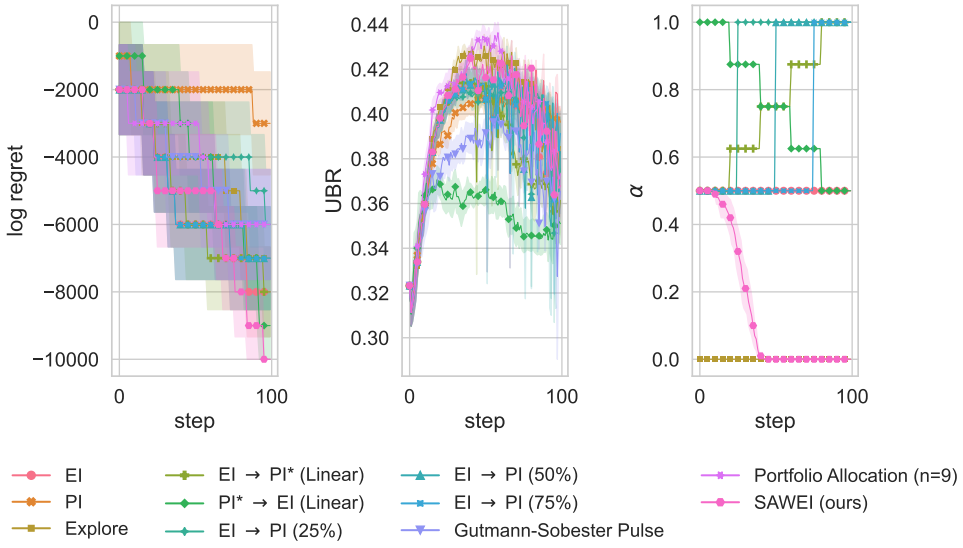


Figure 49: HPOBench ML: (model, task_id) = ('svm', 146818)

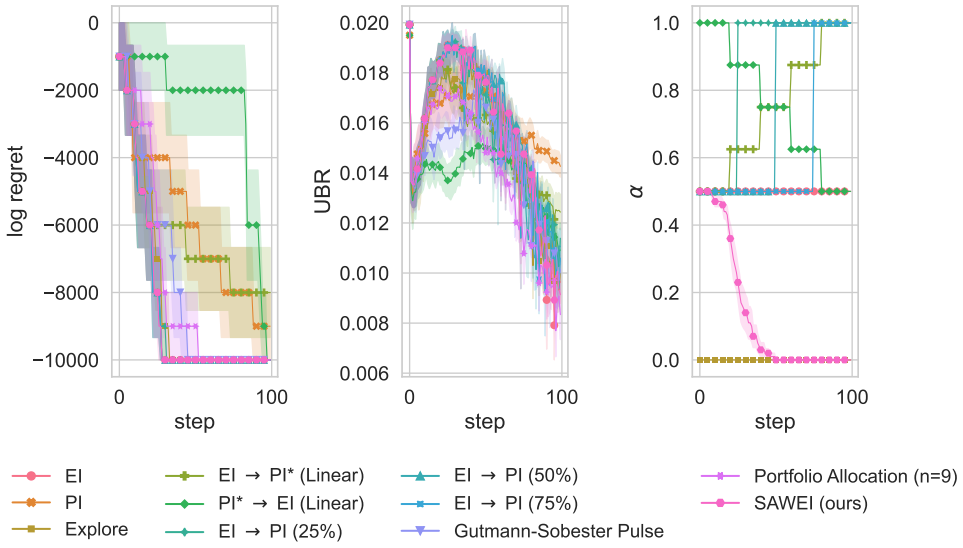


Figure 50: HPOBench ML: (model, task_id) = ('svm', 3917)

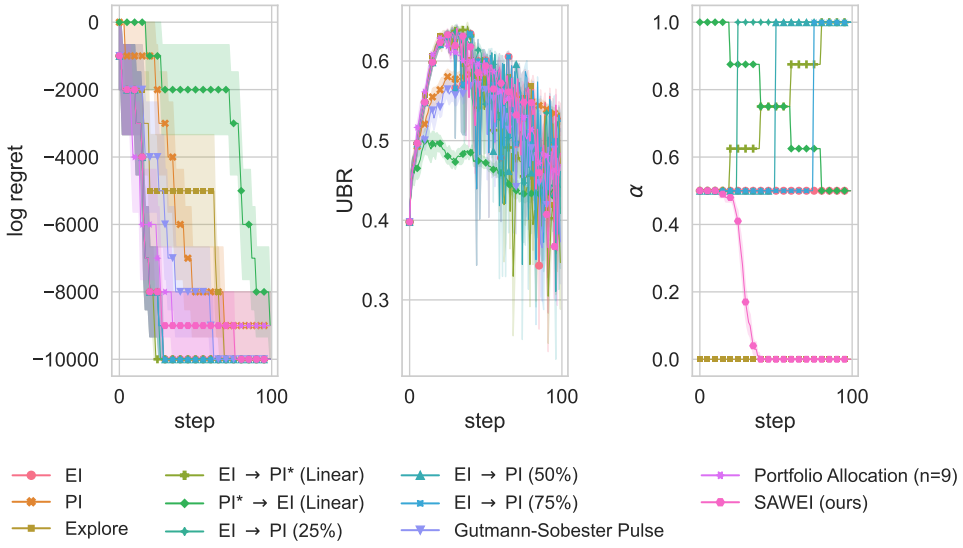


Figure 51: HPOBench ML: (model, task_id) = ('svm', 53)

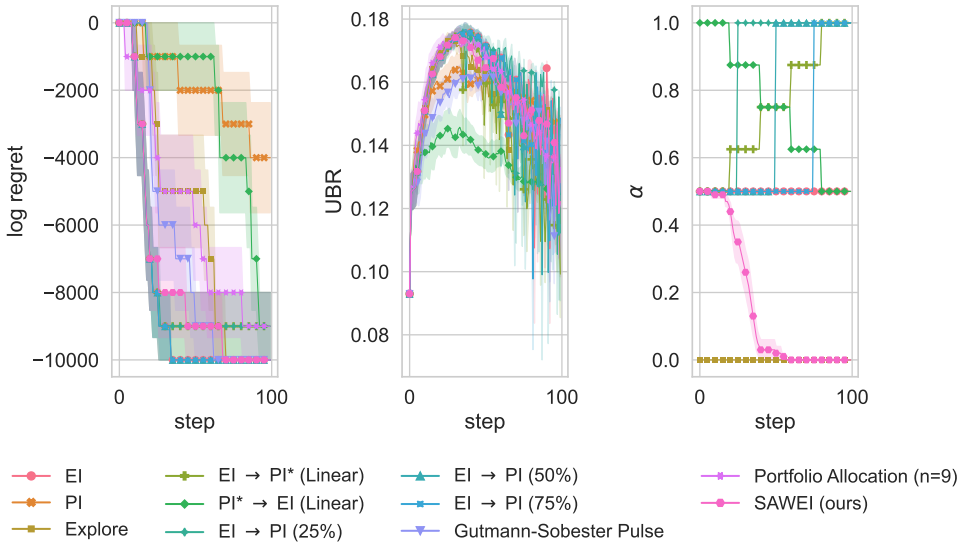


Figure 52: HPOBench ML: (model, task_id) = ('svm', 9952)

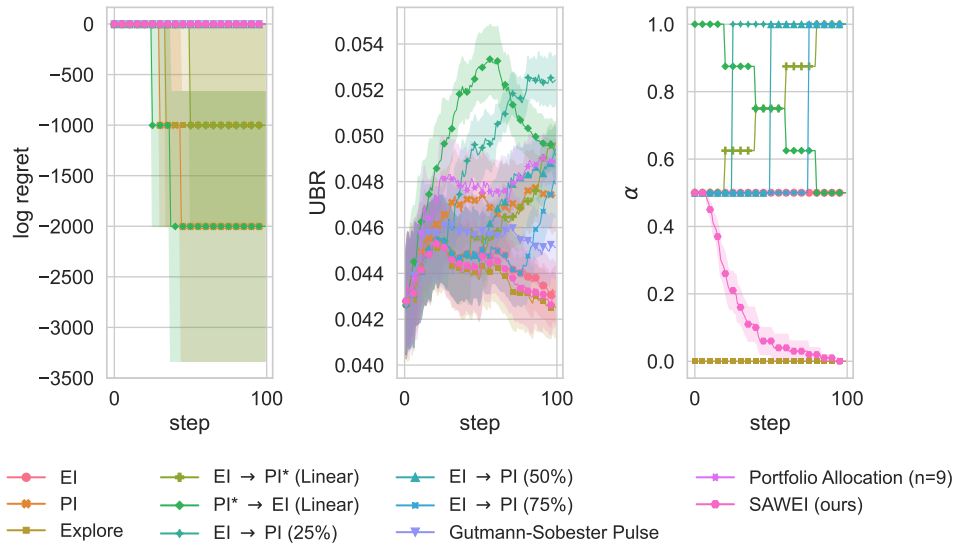


Figure 53: HPOBench ML: (model, task_id) = ('xgb', 10101)

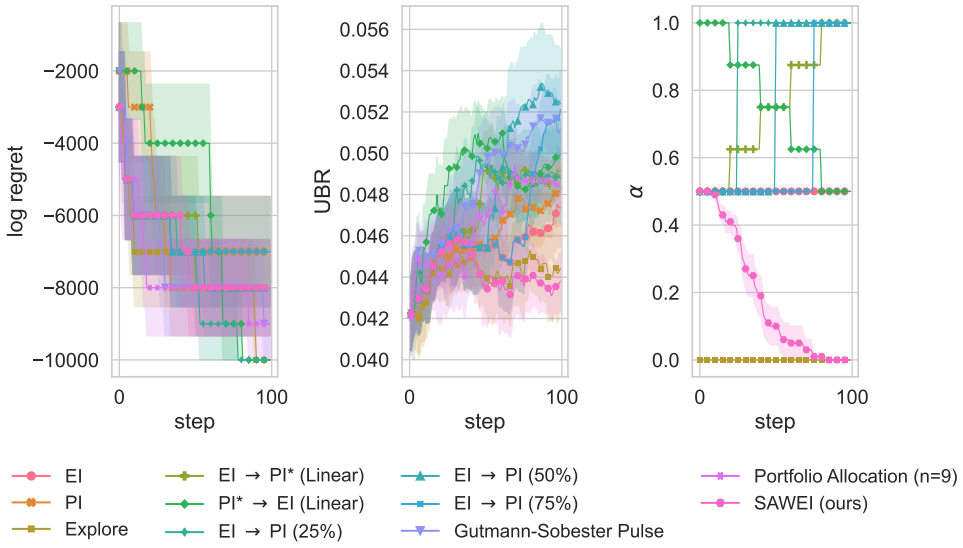


Figure 54: HPOBench ML: (model, task_id) = ('xgb', 146818)

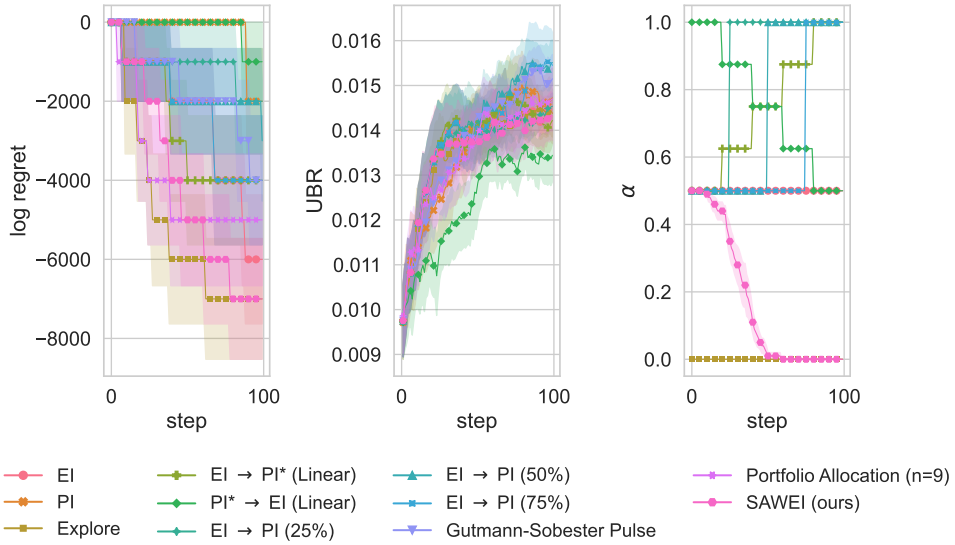


Figure 55: HPOBench ML: (model, task_id) = ('xgb', 3917)

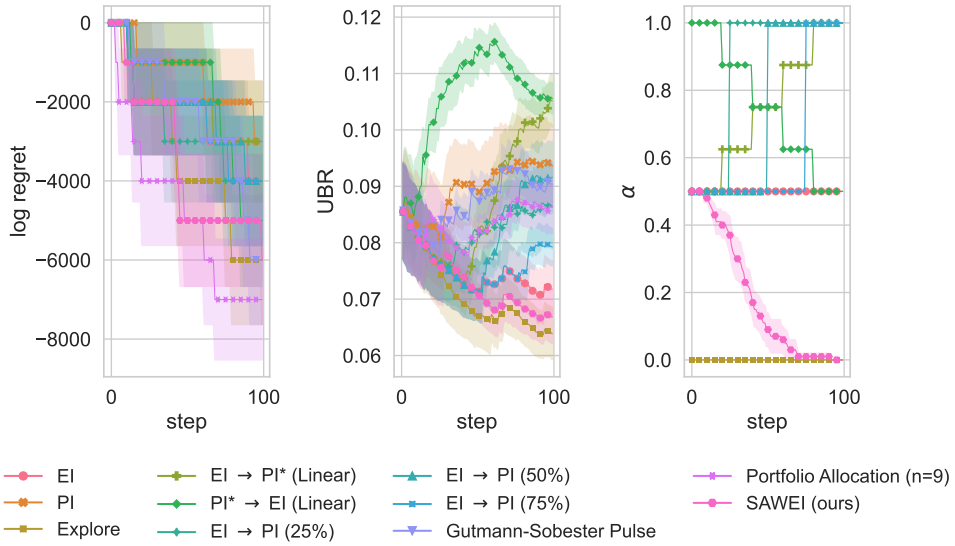


Figure 56: HPOBench ML: (model, task_id) = ('xgb', 53)

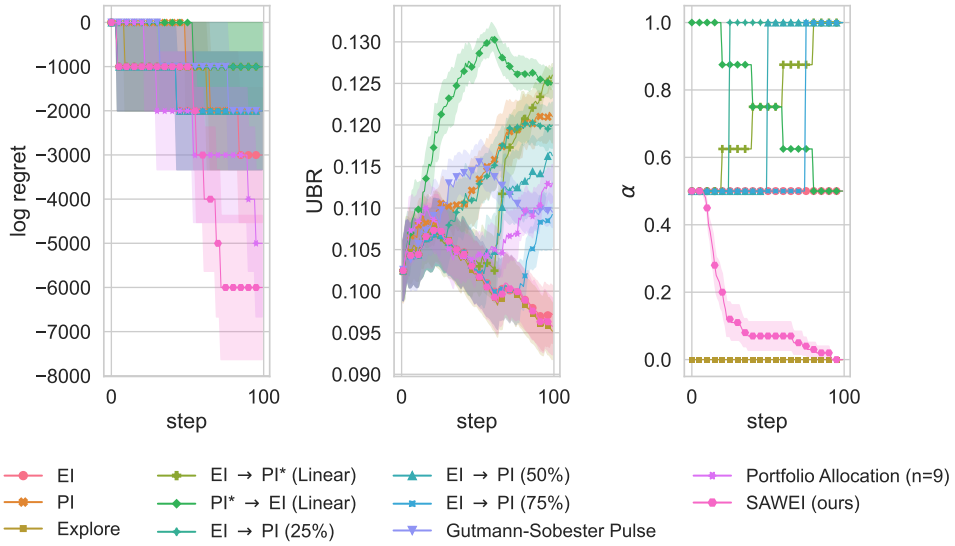


Figure 57: HPOBench ML: (model, task_id) = ('xgb', 9952)