SnAKe: Bayesian Optimization via Pathwise Exploration

Anonymous Author(s)
Affiliation
Address
email

Abstract

Bayesian Optimization is a very effective tool for optimizing expensive black-box functions. Inspired by applications developing and characterizing reaction chemistry using droplet microfluidic reactors, we consider a novel setting where the expense of evaluating the function can increase significantly when making large input changes between iterations. We further assume we are working asynchronously, meaning we have to select new queries before evaluating previous experiments. This paper investigates the problem and introduces 'Sequential Bayesian Optimization via Adaptive Connecting Samples' (SnAKe), which provides a solution by considering large batches of queries and preemptively building optimization paths that minimize input costs. We investigate some convergence properties and empirically show that the algorithm is able to achieve regret similar to classical Bayesian Optimization algorithms in both the synchronous and asynchronous settings, while reducing the input costs significantly. We show the method is robust to the choice of its single hyper-parameter and provide a parameter-free alternative.

1 Introduction

We introduce a method for black-box optimization that keeps input variations as small as possible. A black-box function is expensive to evaluate (with respect to time or resources) and we do not have access to gradients. Classically, black-box optimization finds an optimum by sequentially querying the function. We study a variation of this problem, with two important differences. First, we introduce the idea that large changes in inputs, between iterations, cause the function to become more expensive to evaluate. Second, we do not assume observations are available immediately: delay between querying the function and getting a result leads to asynchronous decision making.

As a motivating example, consider a droplet microfluidic reactor [44] (see Figure 6 in Appendix). In such a reactor, we can quickly pump in chemicals, expose them to certain conditions, and collect the results of our experiments as they leave. However, large changes in temperature mean that the reactions are no longer in steady-state and this makes the evaluations unreliable until the system stabilizes. Smaller changes mean that we never leave steady state, or that the system is easier to stabilize. Further, we have to wait for droplets to exit the reactor before obtaining observations.

More broadly, consider optimization requiring spatially continuous exploration. Black-box optimization has been investigated for optimal placement of pollution sensors [15], and machine learning methods for detecting the sources of pollution have been developed [28]. Mobile pollution readers, such as those used by AirQo in Uganda [`https://www.airqo.net/about`], would face similar challenges if black-box optimizers were used to track the pollution sources. Samaniego et al. [36] show a more concrete example where autonomous vehicles use black-box optimizers to monitor and track water contamination sources in the Ypacarai lake in Paraguay.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

Classical Bayesian Optimization (BO) [18, 37] provides effective solutions to black-box optimization. However, typically BO follows a myopic approach, in that BO chooses the next query based only on the current state of the surrogate model. Classical BO reduces uncertainty in unexplored areas and then quickly returns to promising areas with no regard for the distance between consecutive query points in the input space. This means that BO will incur very high *input costs*.

However, having zero changes in input space is obviously not a good solution. After all we want to explore the search space to find the optimum point. We seek an algorithm that preserves the essence of Bayesian Optimization. Consider a scenario where we know a large number of inputs we want to query. In this case, we could simply order the queries to attain the smallest input cost. A good solution would simply require selecting a large number of queries, creating an ordering, and then simply following the *path* defined by the ordering. Once we start obtaining new information, we could update our beliefs and update our optimization path.

This paper proposes ***S**equ**e**ntial Bayesian Optimization via **A**daptive (**K**)Connecting Sampl**e**s* (SnAKe). Just as the snake grows by carefully eating items in the classic arcade game, a SnAKe optimization path grows from carefully adding queries to the evaluation path.

2 Related Work

'Process-constrained Batch Bayesian Optimization' [45], is the closest analog to our setting of BO with input costs. Vellanki et al. [45] navigate the physical limitations in changing the input space by *fixing* the complicated inputs for every batch. Rather than fixing the difficult-to-change inputs, we *penalize* large input variations in line with the costs (time and resources) of changing conditions. SnAKe decides when to make expensive input changes. Waldron et al. [46] compare the use of transient variable ramps (small input changes in a pre-determined manner) against a full steady-state design of experiments approach for learning parameters of chemical kinetic models. They show that the transient approaches give less precise estimates but much faster. SnAKe combines the best of both worlds, automatically designing experiments while keeping input changes small.

'Cost-aware Bayesian Optimization' [26, 29, 39] optimizes the regret with respect to cost per iteration. These costs, which are fixed throughout the BO process, arise when some regions of the input domain are more expensive to evaluate than others. Similar ideas appear in Multi-fidelity BO [19, 33] where cheap approximations for the objective function are used. Diverging from prior works, our setting does not assume that different regions incur different costs. Instead, SnAKe addresses the setting where the *difference* between adjacent inputs causes the costs to change. The query order changes the cost, therefore we focus on optimization paths instead of optimization sets.

The baseline of Cost-aware BO, Expected Improvement per unit Cost (EIpu), $\mathrm{EI}(x)/\mathrm{Cost}(x)$, is not directly applicable to our setting because the cost of not changing inputs, i.e. choosing $x_t = x_{t-1}$, is zero. We can instead use:

$$\gamma\mathrm{EIpu}(x) = \mathrm{EI}(x)/(\gamma + \mathrm{Cost}(x, x_{t-1})) \tag{1}$$

Three main issues: (a) Eq. (1) introduces new hyper-parameter γ with no obvious way to choose it, (b) Eq. (1) effectively penalizes the acquisition function far away from the data, so near local optima we only penalize exploration and may over-exploit, (c) Asynchronous extensions have a tendency to encourage batch diversity, so we introduce a new trade-off between penalizing locally and far away. As we shall see, SnAKe shows robust results with just one hyper-parameter.

Look ahead Bayesian Optimization [10, 13, 24] considers possible future queries to make our current choice less myopic. This would be a very great way of selecting the inputs to query, as we could simply look-ahead at what any classical BO would choose and then order the queries accordingly. However, one is only able to look ahead for a few iterations, due to the computational complexity of looking too far ahead into the future [5, 48]. For example, [27] combines Markov Decision Processes [34], look-ahead and cost-aware ideas, but is limited by the short time horizons rollout can handle. Reinforcement Learning [43] can be used with longer time-horizons, but it requires access to a simulated environment to train.

A computationally cheaper alternative which allows us to select many inputs to query is Batch Bayesian Optimization (BBO) [2, 12]. BBO is the setting where we are able to parallelize function evaluations, and as such we want to select multiple queries simultaneously. González et al. [13] and Jiang et al. [17] link BBO with look-ahead BO, using a Local Penalization method [12] and Expected

Improvement (q-EI) [11] respectively. Unfortunately, they restrict themselves to smaller batch sizes ($q \leq 15$) due to computational expense. *Asynchronous* Bayesian Optimization [1, 20] addresses the problem of choosing queries while waiting for delayed observations.

3 Methods

3.1 Problem Set-up

We consider finding the maximum, $x^* = \arg\max_{x \in \mathcal{X}} f(x)$, of a black-box function, f, where $\mathcal{X}$ is a compact subset of $\mathbb{R}^d$. We assume f is continuously differentiable and expensive to evaluate. We seek the optimum point while keeping the number of evaluations small. We evaluate the function sequentially, over a discrete and finite number of samples, $t = 1, ..., T$. For every query, x_t, we obtain a noisy observation of the objective, $y_t = f(x_t) + \eta_t$, where $\eta_t \sim \mathcal{N}(0, s^2)$ is Gaussian noise.

We assume there is delay, t_{delay}, between choosing a query and getting an observation. So our data-set at iteration t is given by $D_t = \{(x_i, y_i) : i = 1, ..., t - t_{delay} - 1\}$. If we set $t_{delay} = 0$, we revert to classical sequential Bayesian Optimization, otherwise we are in an *asynchronous* setting.

Finally, we assume there is a *known* inherent cost to changing the inputs to our evaluation, $\mathcal{C}(x_t, x_{t+1})$. We will be using simple regret, $SR_t = f(x^*) - \max_{i=1,...,t} f(x_i)$ as the performance metric. We want to minimize regret while keeping $\sum_{t=1}^{T-1} \mathcal{C}(x_t, x_{t+1})$ small.

As is common in Bayesian Optimization, we will model the black-box function by putting a Gaussian Process (GP) prior on $f \sim \mathcal{GP}(\mu_0, \sigma_0^2)$. Since we have Gaussian noise, the posterior, $f|D_t$ is also a GP, whose mean function, $\mu_t(\cdot)$, and covariance function, $\kappa_t(\cdot, \cdot)$, can be calculated analytically [35].

3.2 General Approach

For our general approach, we will seek to create a large *batch* of queries that we want to evaluate, and then *plan-ahead* a whole optimization path. This is useful for three reasons: (1) it allows us to order the queries in a way that reduces input cost. (2) It allows us to deal with any delay in getting observations, because we can pre-select future queries. (3) It remains computationally feasible to plan-ahead even for very large time horizons. We can follow this ordering or path until new information is available, after which we will update our path.

While we will provide a specific way of creating, planning and updating the paths, we note that the general ideas can be extended to suit different settings. As an example, in Appendix C we show how to alter SnAKe to optimize multiple black-box functions simultaneously.

3.3 Creating a Batch Through Thompson Sampling

We need to produce batches that are representative of the current state of the surrogate model. Not only this, but the method should allow for big batch sizes, since we want to produce batches as big as our budget (which is usually much larger than the batch size most methods consider). For example, in a micro-reactor, we might be interested in batches that contain hundreds of points [44].

Kandasamy et al. [20] offers a promising solution where every point in the batch is independent. The method is based on Thompson Sampling, which uses the GP's own randomness to create a batch. A single batch point is chosen by drawing a realization of the GP, and optimizing it. The queries will fill out the space, and they are more likely to be on promising, and unexplored areas. It should work very well in our context given we expect our initial batch sizes to be very large, so the sample should be representative of the current state of our surrogate model.

3.4 Creating a Path via the Travelling Salesman Problem

After selecting a batch of queries, $\mathcal{P}_t = \{x_t^{(i)}\}_{i=1}^{\tilde{T}}$, we order them. We do this by embedding a graph into the batch, where the edge weights are the cost for changing one input to another. We then find the shortest path that visits every point, i.e., solve the Travelling Salesman Problem (TSP) [4, 6]. Section 3.9 discusses the computational cost.

Mathematically, we define the graph $G = (V,\ E,\ W)$, with $V = \{i \in 1, ..., \tilde{T} : x_t^{(i)} \in \mathcal{P}_t\}$, $E = \{(i,j) : i,j \in 1, ..., \tilde{T}\}$, and $W = \{w_{ij} = \mathcal{C}(x_t^{(i)}, x_t^{(j)}) : (i,j) \in E\}$, where $\tilde{T}$ is the number of batch samples. We solve the TSP in G to obtain our latest optimization path. A simple example would be to try to minimize the total distance travelled in input space, by selecting the Euclidean norm as cost $\mathcal{C}(x_t, x_{t+1}) = ||x_t - x_{t+1}||$.

3.5 Naively Updating the Optimization Path

After updating the GP with new observations, we want to use this information to update our path. We propose updating our strategy by creating a new batch of points.

At iteration t, the remaining budget has size $T - t$. We first propose sampling $T - t$ queries through Thompson Sampling, and then solving the Travelling Salesman Problem. However, we show this leads to the algorithm getting 'stuck' in local optima. This is because every time we re-sample, we naturally include *some exploitation* in the batch, and this exploitation will always be the next point chosen by the TSP - we will never reach the *exploration* algorithm steps. Figure 1a shows an example.

3.6 Escape Analysis

To try and solve the convergence problem introduced by naive resampling, we briefly analyze it. This analysis assumes we receive noise-less observations. However, all sampling can be done in the presence of noise by calculating the corresponding posterior. We focus on Thompson Sampling.

Definition 3.1. (Thompson Sample) We say $x_t^{(i)}$ is Thompson Sampled, and we write $x_t^{(i)} \sim \tau_t$, if:

$$x_t^{(i)} = \arg\max_{x \in \mathcal{X}} f_t^{(i)}(x) \ \text{ where } \ f_t^{(i)}(\cdot) \sim \mathcal{GP}(\mu_t,\ \kappa_t | D_t)$$

Note that the sampling distribution changes at each iteration.

One particular concern, diagrammed in Figure 1a, is the possibility of the method being stuck in a certain area. Let $B_\delta(a)$ be a Euclidean δ-ball centered at a. Assume further that $x_{t-1} \in B_\delta(a)$.

Definition 3.2. (Non-escape Probability) We define the non-escape probability, p_t, at time t, as the probability of a Thompson Sample falling into $B_\delta(a)$. That is, for $x_t^{(i)} \sim \tau_t$:

$$p_t = \mathbb{P}(x_t^{(i)} \in B_\delta(a)) = \mathbb{P}(||x_t^{(i)} - a|| \leq \delta) \tag{2}$$

Let $\mathcal{P}_t = (x_t^{(1)}, ..., x_t^{(T-t)}) \sim \tau_t \ i.i.d.$. Of particular interest to us, is the number of 'non-escapes' in the sample, $N_t = |\mathcal{P}_t \cap B_\delta(a)|$. We are guaranteed to escape if no sample falls in $B_\delta(a)$, so we say we have *fully* escaped if $x_t^{(i)} \notin B_\delta(a) \ \ \forall i$.

Remark 3.3. Note that $N_t \sim$ Binomial$(T - t,\ p_t)$ as all the samples are mutually independent, therefore the probability of fully escaping is $\mathbb{P}(N_t = 0) = (1 - p_t)^{T-t}$. This means we can only expect to fully escape if p_t is very small.

Therefore we are interested in the behavior of p_t as we gain more and more information about f in $B_\delta(a)$. We now consider the circumstances under which p_t becomes very small:

Areas without stationary points: Consider the case when our objective f does not contain a stationary point in $B_\delta(a)$. Then the maximum of f on the closure of the ball, $\bar{B}_\delta(a)$, must lie on the boundary of the ball. In particular, if we assume that our GP model has no error in $B_\delta(a)$, and assuming continuity of sample paths, then it must be $p_t = 0$.

Intuitively, the area itself contains enough information to ensure, with complete certainty, that the global optimum does not lie in the area. We hope that, as we collect information inside areas without stationary points, $p_t \to 0$, and we will eventually leave them with small probability of returning. Appendix D.2 shows an example of this happening very fast.

Areas with stationary points: Areas with stationary points pose a much bigger problem. We will restrict our arguments to local maximums since this is where we have observed the problem. Assume that f has a local optimum in $B_\delta(a)$, higher than any other we have observed before. In this case, any sample taken from a Gaussian Process with no error in $B_\delta(a)$ will have a local maximum inside $B_\delta(a)$, and therefore it is possible that this local maximum is the global solution.

As we increase the information inside the area, p_t is not guaranteed go down to zero. This makes intuitive sense; the only way of knowing if a local optimum is not a global optimum is by sampling away from it - therefore with limited information we will allocate a certain probability to the global optimum being inside $B_\delta(a)$. We include a clear example where $p_t \to p > 0$ in Appendix D.1.

Sampling consistently in a promising area is not necessarily a bad thing, indeed we want to exploit near possible global optimums. However, the question then becomes, how long will it take us to leave? Recall the probability of fully escaping is $(1-p_t)^{T-t}$, and therefore it will be increasing as t increases, even if p_t is (almost) constant.

Remark 3.4. Assume that $p_t \to p > 0$, and that we have a high escape probability after t_e iterations, i.e. $(1-p)^{T-t_e}$ is large, leaving us $T-t_e$ iterations to explore the rest of the space. If we increase our budget from T to T', we will not have a high probability of escape until $(1-p)^{T'-t} = (1-p)^{T-t_e}$, i.e. $t = T' - T + t_e$. This leaves us with $T' - T' + T - t_e = T - t_e$ iterations to explore the rest of the space. Increasing our budget *does not* increase our budget after leaving $B_\delta(a)$! It only means we will be stuck in $B_\delta(a)$ for a longer time. This is very concerning as the method will be very myopic; if it finds a local optimum, it is likely that it will spend a very large amount of the budget exploiting it.

3.7 Escaping with ϵ-Point Deletion

Intuitively, the convergence problem stems from the constant resampling, because every time we re-sample we are reintroducing a number of exploitative points. This forces us to require $N_t = 0$ to be able to fully escape. As a solution, we propose sampling more points than we need to get a measure of the excess exploitation, and then deleting batch points that are similar to previously explored points. Algorithm 1, which we dub ϵ-Point Deletion, still allows us to exploit local optima if we sample many points near them, however, it should eventually move on.

Note that Point Deletion uses the Euclidean norm, and it is independent of the cost function. This is because we are trying to escape local minima of simple regret.

Algorithm 1 ϵ-Point Deletion

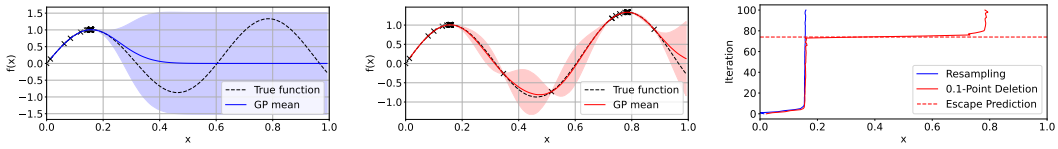
input: New proposed batch $\mathcal{P}_t$ (size T), set of already queried points Q_t (size t), and deletion distance ϵ
for $x \in Q_t$ **do**
  $\tilde{d} \leftarrow \min_{x' \in \mathcal{P}_t} ||x - x'||$
  if $\tilde{d} < \epsilon$ **then**
    `# find the closest point to the query` x `in the new batch`
    $\tilde{x} \leftarrow \arg\min_{x' \in \mathcal{P}_t} ||x - x'||$
  else
    `# else pick a random sample`
    $\tilde{x} \leftarrow \text{Random}(\mathcal{P}_t)$
  end if
  `# remove said point from the batch`
  $\mathcal{P}_t \leftarrow \mathcal{P}_t \setminus \{\tilde{x}\}$
end for
output: A batch $\mathcal{P}_t$ (size $T - t$)

ϵ-Point Deletion allows us to escape local optima by directly increasing the probability of fully escaping without changing p_t. To see this, consider the case where there are q_t previously queried points inside the ball, $B_\delta(a)$, and set $\epsilon \geq 2\delta$. It follows that we will escape if $N_t \geq q_t$, which is much better than requiring $N_t = 0$.

In particular, consider the expected number of 'non-escapes', $\mathbb{E}[N_t] = p_t T$. Around a local maximum, this quantity may be approximately constant with time, $p_t T \approx pT$, so we can reasonably expect an escape when $q_t = pT$. This is a desirable property because we still want to allow exploitation of promising areas. However, this time we leave $T - q_t = T(1-p)$ extra iterations to explore the remaining space! Increasing the budget will benefit *both* the exploitation and the exploration instead of only the former (in contrast with remark 3.4).

Figure 1 gives an empirical example where we use Point Deletion, with $\epsilon = 0.1$, to escape a local optimum. We observe the expected behavior from our brief analysis. For Point Deletion, we calculate

the escape prediction as $pT \approx 74$, using $\hat{p} \approx 0.74$, which we estimated in Appendix D.1. We can see that without Point Deletion, we remain stuck in the first local optimum.



(a) Naively Resampling (b) With Point Deletion (c) Optimization paths.

Figure 1: Effect of Point Deletion for ordering-based BO. (a) and (b) show the maximization objective and the underlying surrogate model, with black crosses representing queries and observations. (c) shows the optimization paths. Naive resampling gets stuck in the local optimum ($x = 0.16$) whereas Point Deletion escapes the local solution. We also show the predicted escape time, Tp (after estimating $p \approx 0.74$ in Appendix D.1), which accurately predicts the algorithm behavior in this simple example.

3.8 SnAKe

Algorithm 2 which we dub 'Sequential Bayesian Optimization via Adaptive Connecting Samples' (SnAKe), combines the ideas of previous sections. Figure 2 diagrams the most important steps of SnAKe. Section 4 develops an effective, parameter-free alternative to the choice of ϵ.

Note there is no requirement for data to be available immediately following querying. If $t_{delay} > 0$, we can simply stick to the latest path. It works without modification on the asynchronous setting. This is vital since we were inspired by chemical experiment design which can exhibit asynchronicity.

Algorithm 2 SnAKe

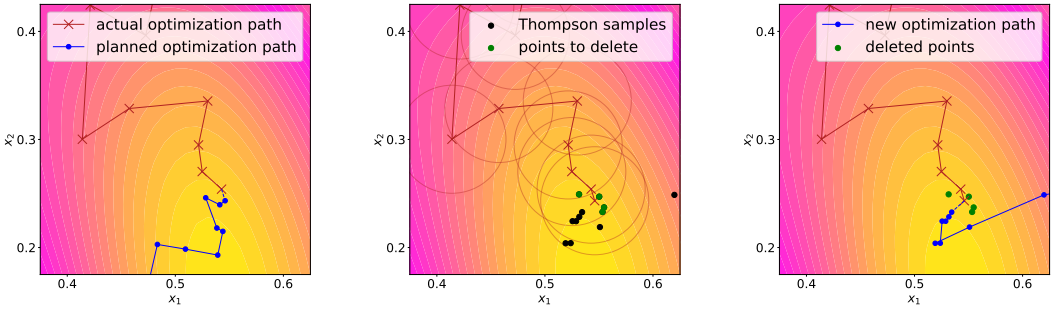
input: Optimization Budget, T. Deletion constant, ϵ.
begin: Create initial batch, $\mathcal{P}_0$, uniformly. Choose starting point x_0. $Q_0 \leftarrow \{x_0\}$. Create initial path, S_0, by solving TSP on $\mathcal{P}_0$.
for $t = 1, 2, 3, ..., T$ **do**
  Check if any running evaluations are finished
  if there are new observations **then**
    Update surrogate model
    Create batch of size T using Thompson Sampling, $\mathcal{P}_{t-1}$
    $\mathcal{P}_{t-1} \leftarrow \epsilon$-Point Deletion($\mathcal{P}_{t-1}$, Q_{t-1})
    $\tilde{S} \leftarrow$ TSP($\mathcal{P}_t$, source = x_{t-1}) $\setminus \{x_{t-1}\}$
    $S \leftarrow Q_{t-1} \cup \tilde{S}$
  end if
  Choose next query point from schedule: $x_t \leftarrow S_t$
  $Q_t \leftarrow Q_{t-1} \cup \{x_t\}$
  Evaluate $f(x_t)$
end for

3.9 Computational Considerations

We use Wilson et al. [47] to efficiently sample the GP, the samples are optimized using Adam [21]. The Travelling Salesman Problem's solution is quickly approximated using Simulated Annealing [22], and we use an adaptive grid to reduce the number of samples away from our current input. With these modifications, we were able to comfortably run all experiments in a CPU (2.5 GHz Quad-Core Intel Core i7). More details can be found in Appendix E.1.

4 Experimental Results

For all experimental results we report the mean and the standard deviation over 25 experimental runs. We give the full implementation details and results in Appendix E and F respectively. All classical BO methods are implemented using BoTorch [3] and GPyTorch [9].

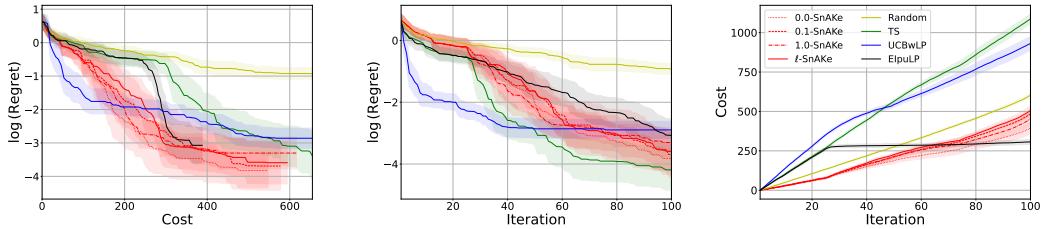


(a) Optimization path at $t = 10$ (b) Sampling & Deletion Step (c) Optimization path at $t = 11$

Figure 2: Graphical example of SnAKe behavior in a full iteration if new information is available. The underlying function is Branin 2D. The feasible set is $[0, 1]^2$, so we do not see all the samples or the complete path with this zoomed-in view. (a) The red line shows the already-queried path. The blue path shows our future plans. (b) For each query, we can see the ϵ-ball under which the deletion step is deterministic. We plot the Thompson Samples as dots, the accepted ones in black and the deleted ones in green. (c) The new path is in blue and the points ignored (due to Point Deletion) are in green. Note the higher concentration of samples in what the model considers a promising area.

In all experiments, we examine SnAKe for $\epsilon = 0, 0.1$, and 1. We further introduce a parameter-free alternative by adaptively selecting ϵ to be the smallest length scale from the GP's kernel, and denote it ℓ-SnAKe. SnAKe proves to be robust to non-zero choices of ϵ, we conjecture this happens because almost all excess exploitation is concentrated on a very small area. For $\epsilon = 0$ we observe very low cost, at the expense of some regret.

Across all experiments we can see how SnAKe can traverse the search space effectively, constantly achieving good regret at low cost. While other cost-aware methods may also achieve good performance, they are very inconsistent suggesting they require careful tuning. SnAKe achieves good, robust results with a single hyper-parameter. We do highlight that SnAKe performs best when there are enough iterations for the samples to adequately fill the search space: this is highlighted by poor performance in Perm10D and by ordinary performance in low-budget experiments (see Appendix F).



(a) Evolution of regret against input cost. We can see that SnAKe is able to achieve the best regret for low cost.

(b) Evolution of regret with iteration number. SnAKe's final regret is comparable with Bayesian Optimization methods, and better than EIpuLP.

(c) Evolution of input cost with iteration number. Low cost is achieved by EIpuLP, SnAKe and the TSP-ordered Random optimization.

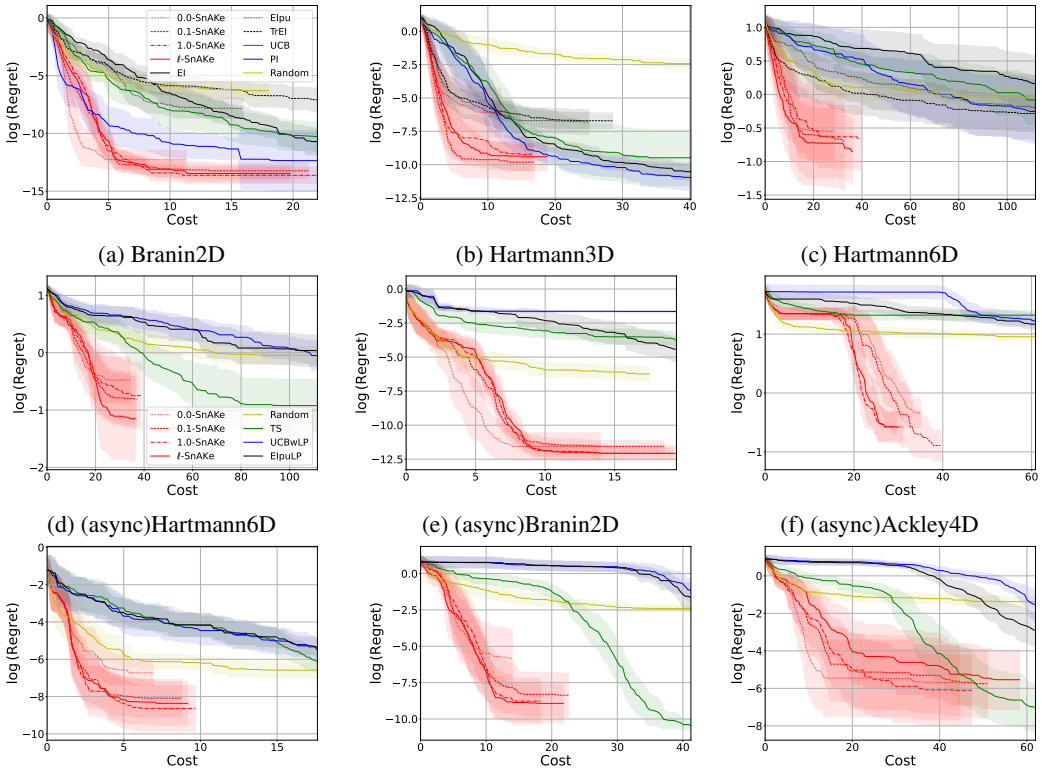
Figure 3: Results of experiments on the SnAr chemistry Benchmark. SnAKe achieves the best regret out of all low-cost methods. The bounds are created from $\pm$ half the standard deviation of all runs. The best performer, ℓ-SnAKe, is parameter-free.

4.1 Synthetic Functions

Sequential BO This section examines the performance of SnAKe against Classical BO algorithms. We compare Expected Improvement (EI) [31], Upper Confidence Bound (UCB) [41], and Probability of Improvement (PI) [23]. We compare against Expected Improvement per Unit Cost [39] as introduced in Eq. (1), with $\gamma = 1$, and with Truncated Expected Improvement [36]. We also introduce a simple baseline, where we create a random Sobol sample, and then arrange an ordering by

solving the TSP, and never update the path again. We do this for three classical benchmark functions, and six total in the Appendix F.1.1. We set the cost function to be the 2-norm distance between the inputs. Figure 4 shows the results in the top row.

Asynchronous BO We explore the asynchronous setting, comparing Local Penalisation with UCB (UCBwLP) [1, 12], Thompson Sampling (TS) [20], and the same Random baseline from the sequential setting. We also test on EIpu with Local Penalisation (EIpuLP), with $\gamma = 1$. The results are shown in last two rows of Figure 4.



(a) Branin2D (b) Hartmann3D (c) Hartmann6D

(d) (async)Hartmann6D (e) (async)Branin2D (f) (async)Ackley4D

(g) (async)Michaelwicz2D (h) (async)Hartmann3D (i) (async)Hartmann4D

Figure 4: Synthetic experimental results. The first row represents three of the synchronous benchmarks. The last two rows correspond to asynchronous experiments with $t_{delay} = 25$. We plot the average $\log(\text{regret})$ achieved against the cost spent. For every experiment, $T = 250$, and we limit the x-axis to the maximum cost achieved by SnAKe or Random. ℓ-SnAKe is the parameter-free version. SnAKe consistently achieves good regret against at cost. The bounds are $\pm$ half the standard deviation of 25 different runs.

4.2 Reaction Control on SnAr Benchmark

We test our method on a real-world, SnAr chemistry benchmark [16]. We control four variables; equivalents of pyrrolidine, concentration of 2,4 dinitrofluorobenenze at reactor inlet, reactor temperature, and residence time. We assume changing temperature, concentration and residence time incur an input cost, owing to the response time required for the reactor to reach a new steady state. We assume the reactor as a first-order dynamic system, where the response to changes in input is given by:

$$(x_s)_i = (x_t)_i + (1 - e^{-s/\alpha_i})(\Delta x_t)_i \tag{3}$$

Where s denotes the time after experiment x_t is finished, $(x_t)_i$ denotes the ith variable of the tth experiment, $(\Delta x_t)_i = (x_{t+1})_i - (x_t)_i$, and α_i is the system time constant. We assume the system reaches steady state once $|(x_s)_i - (x_{t+1})_i| \leq \beta_i$. For input changes smaller than β_i, we assume a linear cost, defined by a parameter γ_i. Combining with the response time from (3):

$$\mathcal{C}_i(x_t, x_{t+1}) = \gamma_i \min\{\beta_i, |\Delta^{(i)} x_t|\} + \max\left\{0, \alpha_i \log\left(|\Delta^{(i)} x_t|/\beta_i\right)\right\} \tag{4}$$

Table 1: Ypacarai experimental results. We present the average regret from 10 runs $\pm$ the standard deviation, multiplied by 10^3. O1, O2, O3 correspond to the different objectives. All runs are terminated after the cost exceeds 10 units (approximately 100 km). SnAKe is the best performer in 2 out of 3 objectives.

	Regret O1	Regret O2	Regret O3
TrEI	0.475 ± 0.596	0.118 ± 0.104	0.076 ± 0.050
EIpu	0.249 ± 0.225	$\mathbf{0.068 \pm 0.073}$	0.071 ± 0.049
SnAKe	$\mathbf{0.074 \pm 0.063}$	0.109 ± 0.186	$\mathbf{0.036 \pm 0.048}$

We assume that we can change variables simultaneously, so the total input cost is simply the longest response time within a given set of input changes. That is, $C(x_t, x_{t+1}) = \max_{i \in I_c} C_i(x_t, x_{t+1})$, where I_c is the index set of the control variables. We implement the simulation using the Summit package [8]. More details can be found in Appendix E.7.6. We set a delay of $t_{delay} = 25$, and optimize for $T = 100$ iterations. The results of the experiment can be seen in Figure 3.

4.3 Finding Contamination Sources in Ypacarai Lake

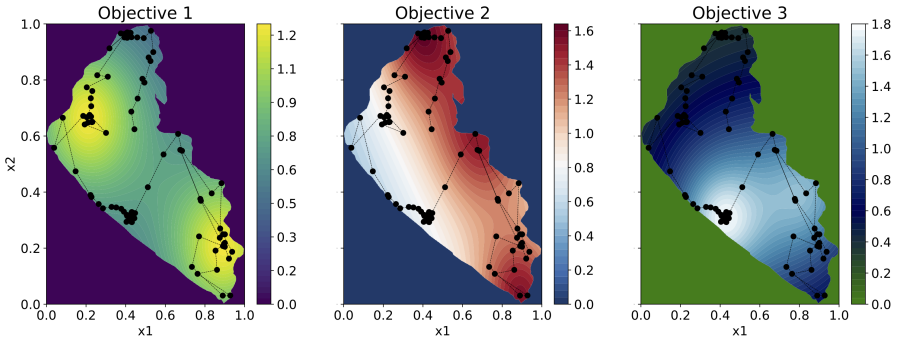


Figure 5: Visualization of Ypacarai Lake, and the corresponding objectives we optimize over. We show an example of an optimization path created using SnAKe for each of the three objectives.

As a second real-world example that exhibits costly input changes, we investigate an alternative to the case study introduced by [36]. Autonomous boats are used to monitor water quality in Ypacarai Lake in Paraguay. We consider three different objectives, illustrated in Figure 5, created using the Schekel benchmark function (as in [36]). Each objective could correspond to a measure of water quality, such as pH, turbidity, CO2 levels, or many more. We consider the problem of finding the *largest* source of contamination in the lake by maximizing these objectives. The cost between queries relates to the boat physically moving the boat from one place to another. We assume that the boats can travel 10 units, which corresponds to approximately 100km (due to fuel limitations), and that observations are available immediately. We run optimization runs on each objective and shows the results in Table 1.

5 Conclusion and Discussion

This paper introduces and proposes a solution to the problem of optimizing black-box functions under costs to changes in inputs. We have empirically shown that the regret achieved by our method is comparable to those of classical Bayesian Optimization methods and we succeed at achieving considerably lower input costs, while being particularly well suited to asynchronous problems.

This setting, with input costs penalizing experimental changes, makes a major step towards automating new reaction chemistry discovery, e.g. in line with the vision of Lazzari et al. [25]. We substantially decrease experimental cost with respect to classical black-box optimization, e.g. as used by Fath et al. [7] and McMullen and Jensen [30]. In the real-life SnAR benchmark, SnAKe spends 45-55% of the cost while making similarly strong predictions to classical BO. The synthetic benchmarks and Ypacarai Lake example offer similar advantages.

References

[1] Ahsan Alvi, Binxin Ru, Jan-Peter Calliess, Stephen Roberts, and Michael A. Osborne. Asynchronous Batch Bayesian Optimisation with Improved Local Penalisation. In *Proceedings of the 36th International Conference on Machine Learning*, pages 253–262, 09–15 Jun 2019.

[2] Javad Azimi, Alan Fern, and Xiaoli Z Fern. Batch Bayesian Optimization via simulation matching. In *Advances in Neural Information Processing Systems*, pages 109–117. Citeseer, 2010.

[3] Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Advances in Neural Information Processing Systems 33*, 2020.

[4] Richard Bellman. Dynamic Programming Treatment of the Travelling Salesman Problem. *Journal of the ACM (JACM)*, 9(1):61–63, 1962.

[5] Dimitri Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, 2012.

[6] Marco Dorigo and Luca Maria Gambardella. Ant colonies for the travelling salesman problem. *Biosystems*, 43(2):73–81, 1997.

[7] Verena Fath, Philipp Lau, Christoph Greve, Philipp Weller, Norbert Kockmann, and Thorsten Röder. Simultaneous self-optimisation of yield and purity through successive combination of inline FT-IR spectroscopy and online mass spectrometry in flow reactions. *Journal of Flow Chemistry*, 11(3):285–302, 2021.

[8] Kobi Felton, Jan Rittig, and Alexei Lapkin. Summit: Benchmarking Machine Learning Methods for Reaction Optimisation. *Chemistry Methods*, February 2021.

[9] Jacob R. Gardner, Geoff Pleiss, David Bindel, Kilian Q. Weinberger, and Andrew Gordon Wilson. Gpytorch: Blackbox matrix-matrix Gaussian Process inference with GPU acceleration. *Advances in Neural Information Processing Systems*, pages 7576–7586, 2018.

[10] David Ginsbourger and Rodolphe Le Riche. Towards Gaussian process-based optimization with finite time horizon. In *Advances in Model-Oriented Design and Analysis*, pages 89–96. Springer, 2010.

[11] David Ginsbourger, Rodolphe Le Riche, and Laurent Carraro. Kriging is well-suited to Parallelize Optimization. In *Computational Intelligence in Expensive Optimization Problems*, Springer series in Evolutionary Learning and Optimization, pages 131–162. 2010.

[12] Javier González, Zhenwen Dai, Philipp Hennig, and Neil Lawrence. Batch Bayesian Optimization via Local Penalization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 648–657, 09–11 May 2016.

[13] Javier González, Michael Osborne, and Neil Lawrence. GLASSES: Relieving The Myopia Of Bayesian Optimisation. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 790–799, 09–11 May 2016.

[14] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using NetworkX. Technical report, Los Alamos National Lab. (LANL), 2008.

[15] Sigrid Passano Hellan, Christopher G Lucas, and Nigel H Goddard. Bayesian Optimisation for Active Monitoring of Air Pollution. In *36th AAAI Conference on Artificial Intelligence*, 2021.

[16] Christopher A. Hone, Nicholas Holmes, Geoffrey R. Akien, Richard A. Bourne, and Frans L. Muller. Rapid multistep kinetic model generation from transient flow data. *React. Chem. Eng.*, 2:103–108, 2017.

[17] Shali Jiang, Henry Chai, Javier González, and Roman Garnett. BINOCULARS for Efficient, Non-myopic Sequential Experimental Design. In *Proceedings of the 37th International Conference on Machine Learning*, pages 4794–4803, 13–18 Jul 2020.

[18] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[19] Kirthevasan Kandasamy, Gautam Dasarathy, Junier B Oliva, Jeff Schneider, and Barnabás Póczos. Gaussian Process Bandit Optimisation with Multi-fidelity Evaluations. *Advances in neural information processing systems*, 29, 2016.

[20] Kirthevasan Kandasamy, Akshay Krishnamurthy, Jeff Schneider, and Barnabas Poczos. Parallelised Bayesian Optimisation via Thompson Sampling. In *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, pages 133–142, 2018.

[21] Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*, 12 2014.

[22] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[23] Harold J. Kushner. A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise. *Journal of Basic Engineering*, 86:97–106, 1964.

[24] Remi Lam, Karen Willcox, and David H Wolpert. Bayesian optimization with a finite budget: An approximate dynamic programming approach. *Advances in Neural Information Processing Systems*, 29:883–891, 2016.

[25] Stefano Lazzari, Andree Lischewski, Yury Orlov, Peter Deglmann, Andreas Daiss, Eduard Schreiner, and Hugo Vale. *Toward a digital polymer reaction engineering*, volume 56, pages 187–227. 01 2020.

[26] Eric Hans Lee, Valerio Perrone, Cedric Archambeau, and Matthias Seeger. Cost-aware Bayesian optimization. *Proceedings of the 7th ICML Workshop on Automated Machine Learning*, 2020.

[27] Eric Hans Lee, David Eriksson, Valerio Perrone, and Matthias Seeger. A Nonmyopic Approach to Cost-Constrained Bayesian Optimization. In *Uncertainty in Artificial Intelligence*, pages 568–577. PMLR, 2021.

[28] Fang-Yie Leu and Jia-Sheng Ho. Air Pollution Source Identification by Using Neural Network with Bayesian Optimization. In *International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 514–524. Springer, 2019.

[29] Phuc Luong, Dang Nguyen, Sunil Gupta, Santu Rana, and Svetha Venkatesh. Adaptive cost-aware Bayesian optimization. *Knowledge-Based Systems*, 232:107481, 2021.

[30] Jonathan P. McMullen and Klavs F. Jensen. Rapid Determination of Reaction Kinetics with an Automated Microfluidic System. *Organic Process Research & Development*, 15(2):398–407, 2011.

[31] J. Mockus, Vytautas Tiesis, and Antanas Zilinskas. The application of Bayesian methods for seeking the extremum. *Towards Global Optimization*, 2:117–129, 09 2014.

[32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[33] Matthias Poloczek, Jialei Wang, and Peter Frazier. Multi-information source optimization. *Advances In Neural Information Processing Systems*, 30, 2017.

[34] Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[35] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[36] Federico Peralta Samaniego, Daniel Gutiérrez Reina, Sergio L Toral Marín, Mario Arzamendia, and Derlis O Gregor. A Bayesian Optimization Approach for Water Resources Monitoring Through An Autonomous Surface Vehicle: The Ypacarai Lake Case Study. *IEEE Access*, 9: 9163–9179, 2021.

[37] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[38] Edward Snelson and Zoubin Ghahramani. Sparse Gaussian Processes Using Pseudo-Inputs. In *Advances in Neural Information Processing Systems*, pages 1257–1264. MIT Press, 2005.

[39] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

[40] Il'ya Meerovich Sobol'. On the distribution of points in a cube and the approximate evaluation of integrals. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, 7(4):784–802, 1967.

[41] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pages 1015–1022, 2010.

[42] S. Surjanovic and D. Bingham. Virtual Library of Simulation Experiments: Test Functions and Datasets. Available at `http://www.sfu.ca/~ssurjano`, 2013.

[43] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[44] Shia-Yen Teh, Robert Lin, Lung-Hsin Hung, and Abraham P. Lee. Droplet microfluidics. *Lab Chip*, 8:198–220, 2008.

[45] Pratibha Vellanki, Santu Rana, Sunil Gupta, David Rubin, Alessandra Sutti, Thomas Dorin, Murray Height, Paul Sanders, and Svetha Venkatesh. Process-constrained Batch Bayesian Optimisation. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017.

[46] Conor Waldron, Arun Pankajakshan, Marco Quaglio, Enhong Cao, Federico Galvanin, and Asterios Gavriilidis. An autonomous microreactor platform for the rapid identification of kinetic models. *React. Chem. Eng.*, 4:1623–1636, 2019.

[47] James Wilson, Viacheslav Borovitskiy, Alexander Terenin, Peter Mostowsky, and Marc Deisenroth. Efficiently Sampling Functions from Gaussian Process Posteriors. In *Proceedings of the 37th International Conference on Machine Learning*, pages 10292–10302, 13-18 Jul 2020.

[48] Xubo Yue and Raed AL Kontar. Why Non-myopic Bayesian Optimization is Promising and How Far Should We Look-ahead? A Study via Rollout. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, pages 2808–2818, 26–28 Aug 2020.

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] We believe the claims in the abstract are backed by the paper's content.
   (b) Did you describe the limitations of your work? [Yes] See the last paragraph of Section 4's introduction.
   (c) Did you discuss any potential negative societal impacts of your work? [No] We do not believe there are any obvious negative societal impacts.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A] See below.
   (b) Did you include complete proofs of all theoretical results? [No] While we include some mathematical analysis in Section 3.6, it is not fully rigorous and it is only meant to illustrate the importance of the Point Deletion heuristic.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The code is included in supplemental material, and will be made public after peer-review. We also give details about how the random seeds for each run were chosen in the appendix.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section E.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Section 4.
   (d) 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] See Section 3.9
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? [Yes] See Section E.
   (b) Did you mention the license of the assets? [No] The licenses can be found in the documentation of each asset.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No] But the GitHub repository URL to generate the paper's result will be included after peer-review.
   (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 Motivating Example

Figure 6 shows a graphic of a droplet microfluidic reactor [44] which is our main motivating example.

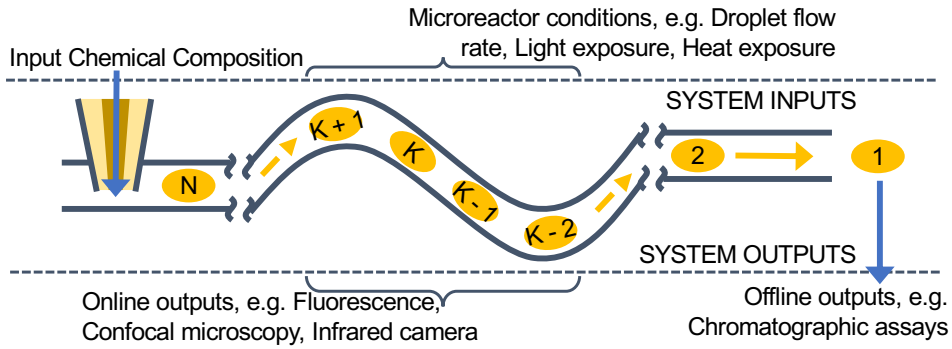


Figure 6: Motivating example. Droplets flow into the micro-reactor where we control conditions such as temperature and flow rate. The cost to change function inputs arises from how adjacent droplets are coupled, e.g. rapidly changing the temperature after droplet K means waiting for system equilibration before taking new measurements. Asynchronicity arises from choosing drops $2, 3, ..., N$ before getting the results of droplet 1.

B General Approach

While we present SnAKe as a detailed algorithm, the ideas behind it are far more general. For example, in Section C we create our batch by taking samples from three different GPs. Algorithm 3 presents a more general algorithm that could be used for any similar task.

Algorithm 3 General Ordering-Based Optimization

input: Optimization budget of T samples. Method for creating batch of queries. Method for creating an ordering from a batch of queries. Method for updating paths.
begin: Create initial batch of size T and ordering, S
for $t = 1, 2, 3, ..., T$ **do**
  if there is new information **then**
    Update surrogate model
    Choose a batch of new points to query
    Create a new path, $\tilde{S}$
    $S \leftarrow \tilde{S}$
  end if
  Choose next query point from ordering: $x_t \leftarrow S_t$
  Evaluate $f(x_t)$
end for

C Ypacarai Lake Experiment

A second motivating example for our research comes from optimization requiring spatially continuous exploration. In this section, we examine the problem more closely, and show that SnAKe can easily be extended to optimize multiple black-box functions *simultaneously* in the same search space.

The problem in question is an adaptation of the case study introduced by Samaniego et al. [36]. The application is to use autonomous boats to find the *largest* source of contamination in Lake Ypacarai in Paraguay. Figure 7 shows a visualization of the Lake, and three objectives we want to optimize over. Each objective corresponds to a different measure of water contamination, examples might include pH, turbidity, CO2 levels, and more. In Section 4.3 we explored optimizing over these examples individually.

Table 2: Ypacarai experimental results. We present the average regret from 10 runs $\pm$ the standard deviation multiplied by 10^3. MO1, MO2, MO3 represent the different maximums (one for each function). All runs are terminated after the cost exceeds 10 units (approximately 100 km). SnAKe is the only method with good regret in all metrics, as TrEI performs poorly in the first objective, and EIpu performs poorly in the second.

	Regret MO1	Regret MO2	Regret MO3
TrEI	13.636 ± 35.842	0.802 ± 0.828	0.162 ± 0.108
EIpu	0.995 ± 0.661	19.308 ± 55.901	0.212 ± 0.137
SnAKe	0.996 ± 2.743	1.740 ± 4.813	0.064 ± 0.053

In this section, we extend the problem to optimize over all of these *simultaneously*. This is important because it is inefficient to run multiple objective runs when we are exploring the same search space. At every iteration we choose one query x_t, and obtain three outputs $f_1(x_t)$, $f_2(x_t)$, and $f_3(x_t)$ one for each objective.

We model each objective using an independent GP, meaning we can *create Thompson Samples from any objective*. Therefore we extend SnAKe by creating the batch using a mixture of samples from each objective. For the experiment we use the ratio 1:1:1. For TrEI and EIpu, we adapt to the multi-objective case by optimizing the first objective until we have travelled 3.3 units (approximately 33km), after which we change to the subsequent objective. Table 2 shows the results.

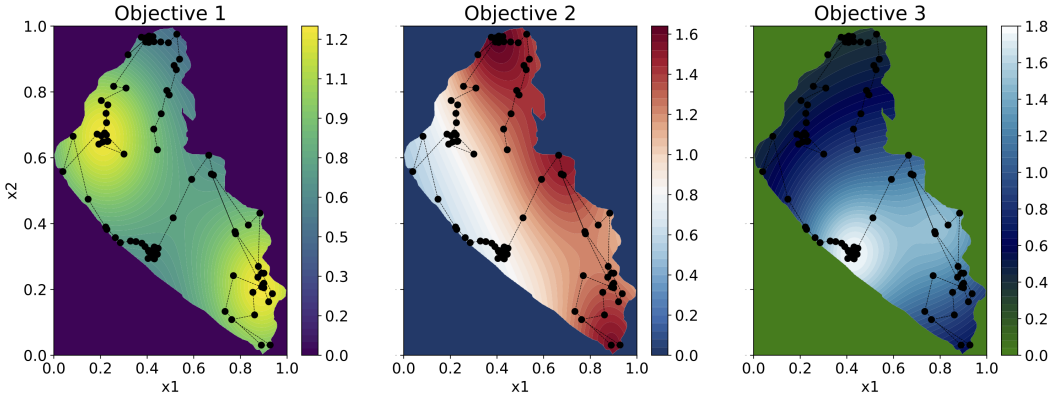


Figure 7: Visualization of Ypacarai Lake, and the corresponding objectives we optimize over. We show an example of an optimization path.

D Empirical Analysis of Escape Probability

D.1 Areas with stationary points

Figure 8 estimates the non-escape probability (see Definition 3.2) from the interval $A = [0.1, 0.2]$. The optimization objective is a bi-modal function. Once we have 15 samples in the interval $[0.1, 0.2]$, we estimate the escape probability to converge to $p \approx 0.74$.

D.2 Areas without stationary points

We now repeat the same experiment as in section D.1, this time we change the interval to $A = [0.0, 0.1]$ which does not contain any stationary points. One can observe a clear difference in the behavior of p_t as we include more information. This time, $p_t \to 0$ very fast.

D.3 Resampling vs Point Deletion

Figure 10 empirically confirms the analysis of Section 3.7 showing the effectiveness of ϵ-Point Deletion, and displaying the effect of increasing the budget from 100 to 250 iterations.

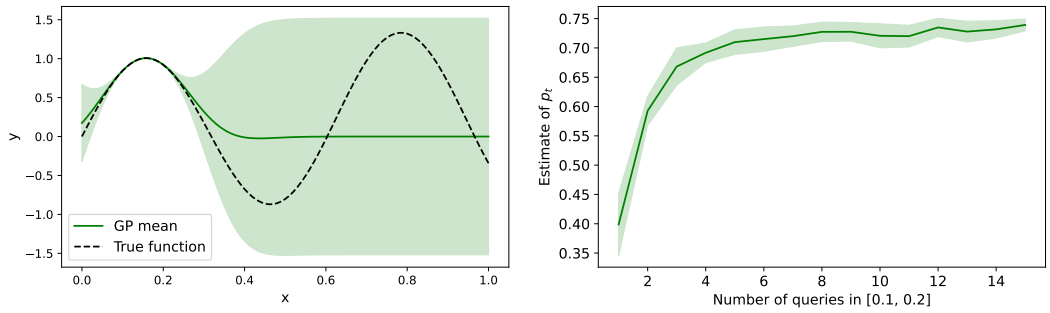


Figure 8: We estimate the probability of non-escape by taking 5000 independent Thompson Samples and counting the number of samples inside A (i.e. the MLE estimator of the Bernoulli distribution). We do this for increasing number of training points in A (which are chosen randomly with a uniform distribution in A). We repeat the experiment 10 times. The left plot shows the underlying function and the Gaussian Process for 15 training points. The right plot shows the evolution of our estimate as we increase training points inside $A = [0.1, 0.2]$ (we plot the mean of each run $\pm$ the standard deviation). This example makes it clear that p_t does not converge to zero. Furthermore, it seems to converge to just over 0.7 which a very large probability. This will make fully escaping the local minimum very difficult without Point Deletion.

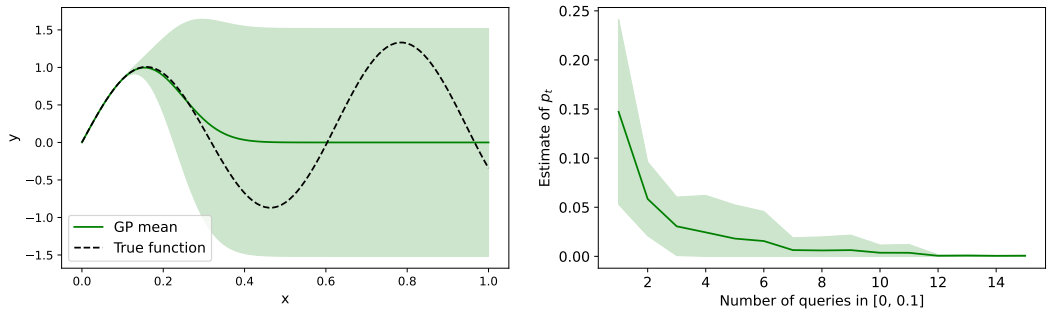


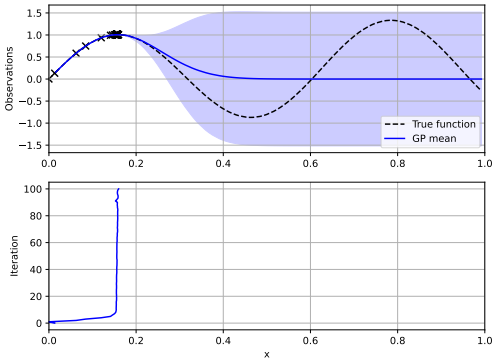
Figure 9: We estimate the probability of non-escape by taking 5000 independent Thompson Samples and counting the number of samples inside A (i.e. the MLE estimator of the Bernoulli distribution). We do this for increasing number of training points in A (which are chosen randomly with a uniform distribution in A). We repeat the experiment 10 times. The left plot shows the underlying function and the Gaussian Process for 15 training points. The right plot shows the evolution of our estimate as we increase training points inside $A = [0, 0.1]$ (we plot the mean of each run $\pm$ the standard deviation). We can see that p_t quickly converges to (almost) zero. We almost guaranteed to fully escape the area after 15 time-steps, even for very large budgets.

E Implementation Details

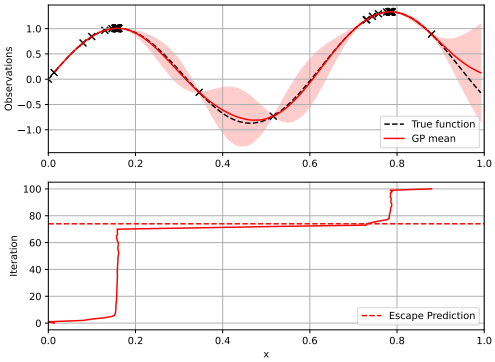
This section outlines the implementation details and hyper-parameter choices for all the methods compared in the paper. The code will be made available open source after peer review.

E.1 Computational Considerations (Extended)

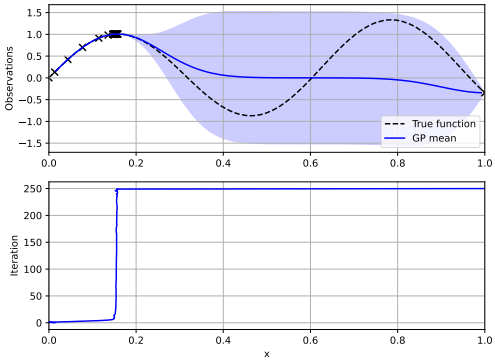
Unfortunately, Algorithm 2 (SnAKe) is computationally expensive in two aspects. First, for large budgets, we may struggle to train and sample the GPs. For our experiments, training was not an issue and we were able to use full model GPs. However, we could use Sparse GPs [38] if needed. The Wilson et al. [47] approach allows us to create GP samples efficiently, and possibly optimize them using gradient methods. The sampling can be done in linear time (*after* the GP has been trained). We use the Wilson et al. [47] method to create our samples, and then optimize the samples using Adam [21].



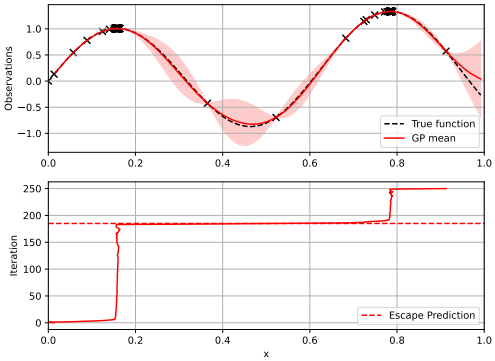
(a) Optimization with Resampling. $T = 100$.



(b) Optimization with 0.1-Point Deletion. $T = 100$.



(c) Optimization with Resampling. $T = 250$.



(d) Optimization with 0.1-Point Deletion. $T = 250$.

Figure 10: We investigate the effect of Point Deletion, and empirically confirm our analysis from Section 3.6. For Point Deletion, we calculate the escape prediction as pT, using $\hat{p} \approx 0.74$, which we estimated in Section D.1. We can see that without Point Deletion, the escape happens until *until the very last* iteration, independently of the budget (see Remark 3.4 for an explanation). With Point Deletion, we can see that our escape predictions are accurate, and the exploration of the actual optimum increases with the budget.

The second bottleneck is solving the TSP, which is NP-hard, and we may need to solve it almost at every iteration (for small values of t_{delay}). There are heuristic solutions that give approximate solutions quickly. We use Simulated Annealing [22] which grows linearly with the budget size, T. For small budgets, simulated annealing should find good solutions, but it could struggle as the budget size increases. To solve this, note that we do not actually require a super-specific solution to the problem: we are only expecting to query the first few points on a path before replacing it by an entirely new path.

We build an adaptive grid (at each iteration) consisting of two separate parts. A very coarse grid, ξ_{global}, covers most of the search space, and a very fine grid, $\xi_{local,t}$, consisting of the N_l samples closest to x_t. This allows us to define the grid $\xi_t = \xi_{global} \cup \xi_{local,t}$. The remaining $T - N_l$ samples will be assigned to the closest point in ξ_t (using the Euclidean distance).

The adaptive grid means we expect to have multiple samples assigned to the same point, specifically in the coarse areas of our grid. But this is not important, because we expect our immediate attention will be in the area around our current input where there should be little to no repetition. This will allow the algorithm to focus on testing solutions which are relevant to our problem.

The adaptive grid introduces two hyper-parameters: the size of the global and local grids, respectively N_g and N_l. Using this method, we run the TSP heuristics on a graph with at most $\min(N_g + N_l, T)$ nodes. For the experiments, we create the global grid using a simple Sobol grid [40].

E.2 Gaussian Processes

For every GP, we use the RBF Kernel with an output-scale, θ_0:

$$k_{RBF}(x_1, x_2) = \theta_0 \exp\left(-\frac{1}{2}(x_1 - x_2)^T \Theta^{-2}(x_1 - x_2)\right)$$

where $\Theta = \text{diag}(\ell_1, ..., \ell_d)$ and ℓ_i denotes the length-scale of the ith variable. For the prior mean, we used a constant function with trainable value, μ_0. We implemented them all using the package GPyTorch [9].

E.3 Training the hyper-parameters of the Gaussian Processes

Our method is well suited for physical systems. Hence, we assume that there is good prior knowledge of the hyper-parameters. In particular, we found it reasonable that each hyper-parameter would be given a lower and upper bound. Normally, we would simply have a large initialization sample. However, we believe this goes against the nature of the problem because we want to explore the space slowly to avoid large input costs. So any type of initialization would be costly.

We simulate this in the following way: we first randomly sample $\max(T/5, 10d)$ points from the d-dimensional input space, and train a GP on these data-points. The hyper-parameters of the GP are optimized by maximizing the marginal log-likelihood [35] over 500 epochs using Adam [21] with a learning rate of 0.01. The resulting hyper-parameters will correspond to the 'educated guess'. We then set the following bounds:

a) For the length-scale, the lower bound is half the educated guess, and the upper bound double the 'educated guess'.

b) For the output-scale, θ_0, the lower bound is half the educated guess, and the upper bound double the 'educated guess'.

c) For the initial mean, μ_0, the lower bound is the educated guess minus a third of the initial variance, the upper bound is the educated guess plus a third of the initial variance.

d) The noise parameter we simply set to be greater than 10^{-5}.

We (partially) enforce the constraints by setting a SmoothedBoxPrior on each parameter, with a variance of 0.001. Finally, under the constraints defined above, we re-estimate the hyper-parameters every time we obtain 25 new observations.

To make sure all models receive fair initializations, we set the same seed for each run and function pair.

E.4 SnAKe

We used Simulated Annealing [22] to solve the Travelling Salesman Problem. We implemented it using the NetworkX package [14]. We initalized the cycle with the 'greedy' sub-algorithm and used all default options.

We generated the Thompson Samples using the method introduced in Wilson et al. [47] which we implemented ourselves. To optimize the samples we used Adam [21] and PyTorch [32] over $10d$ epochs, with a learning rate of 0.01. We used $10d$ multi-starts for each sample. To create the samples, we used $\ell = 1024$ Fourier bases.

For ℓ-SnAKe, we define an adaptive deletion constant, $\epsilon_t = \min(\ell_{1,t}, ..., \ell_{d,t})$, where $\ell_{i,t}$ denotes the length scale of the ith variable at time t (recall we are re-training the hyper-parameters every new 25 observations, so the length scales change with time).

For the adaptive grid, we use $N_l = 25$ local samples, and a corse global Sobol grid [40] of $N_g = 100$ points.

E.5 Classical Bayesian Optimization

We used BoTorch [3] to implement all methods in this section. We optimized the acquisition functions across 150 epochs using Adam [21] with a learning rate of 0.0001 using 7500 random multi-starts.

E.5.1 Expected Improvement

Expected Improvement [31] optimizes the acquisition function:

$$EI(x) = \mathbb{E}\left[\max(y - y_{best}, 0)\right], \quad y \sim f(x)$$

where y_{best} is our best observation so far.

Expected Improvement per Unit Cost [39] optimizes the acquisition function:

$$EI(x, x_{t-1}) = \frac{EI(x))}{C(x, x_{t-1}) + \gamma}$$

We set $\gamma = 1$ in all experiments to address the fact that not moving incurs zero cost. Note that as $\gamma \to \infty$, the method will behave closer to normal Expected Improvement, and as $\gamma \to 0$ the method will tend to stay closer and closer to the current input.

Truncated Expected Improvement [36] first maximizes the normal Expected Improvement acquisition function. It then travels a distance of at most ℓ towards the proposed point, where ℓ is the GPs length-scale. We exclude this method from synchronous SnAr, as it is not suited for general cost functions.

E.5.2 Upper Confidence Bound

Upper Confidence Bound [41] optimizes the acquisition function:

$$UCB(x) = \mu_t(x) + \beta_t \sigma_t(x)$$

We set $\beta_t = 0.2d\log(2t)$ following Kandasamy et al. [19].

E.5.3 Probability of Improvement

Probability of Improvement [23] optimizes the acquisition function:

$$PI(x) = \mathbb{P}(y \geq y_{best}), \quad y \sim f(x)$$

E.5.4 Truncated Expected Improvement

Truncated Expected Improvement was developed by [36] trying to solve the task of using an automated boat to monitor the water quality of Ypacari Lake in Paraguay. The method seeks to take distance travelled into account when doing Bayesian Optimization. It does by selecting a point using Expected Improvement, setting a direct path from our current input to the new point, and then *truncating* the path one length-scale away. That is:

$$p_t = \arg\max_{x \in \mathcal{X}} EI(x \mid D_t)$$
$$x_{t+1} = x_t + \frac{p_t - x_t}{||p_t - x_t||} \min(\ell, ||p_t - x_t||)$$

E.6 Asynchronous Bayesian Optimization

E.6.1 Local Penalization

We use the penalization method as is described in González et al. [12]. For the Lipschitz constant, we estimate it by calculating the gradient of μ_t (using auto-differentiation) in a Sobol grid [40] of $50d$ points and selecting L to be the maximum gradient in the grid.

UCB with Local Penalization We set $\beta_t = 0.2d\log(2t)$ and $M = y_{best}$.

EIpu with Local Penalization We set $\gamma = 1$.

E.6.2 Thompson Sampling

We use the sample procedure as in E.4, except we only optimize a single sample at every iteration.

E.7 Description of Benchmark Functions

We chose the benchmark functions to observe the behavior of SnAKe in a variety of scenarios. More details of all the benchmark functions can be found in Surjanovic and Bingham [42].

E.7.1 Branin2D

The two-dimensional Branin function is given by. The function has three global maximums:

$$f(x) = a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1-t)\cos(x_1) + s$$

where we optimize over $\mathcal{X} = [-5, 10] \times [0, 15]$. We set $a = -1$, $= 5.1/(4\pi^2)$, $c = 5/\pi$, $r = 6$, $s = -10$, and $t = 1/(8\pi)$.

E.7.2 Ackley4D

The four-dimensional Ackley function has a lot of local optimums, with the optima in the center of the search space. The function is given by:

$$f(x) = a\exp\left(-b\sqrt{\frac{1}{4}\sum_{i=1}^{4} x_i^2}\right) + \exp\left(\frac{1}{d}\sum_{i=1}^{4}\cos(cx_i)\right) - a - \exp(1)$$

where we slightly shift the search space and optimize over $\mathcal{X} = [-1.8, 2.2]^4$. This is to avoid having the optimum exactly at a point in the Sobol grid and giving SnAKe an unfair advantage. We set $a = 20$, $= 0.2$, and $c = 2\pi$.

E.7.3 Michaelwicz2D

The two-dimensional Michalewicz function is characterized by multiple local maxima and a lot of flat regions. The function is given by:

$$f(x) = \sum_{i=1}^{2}\sin(x_i)\sin^{2m}\left(\frac{ix_i^2}{\pi}\right)$$

where we set $m = 10$ and we optimize on the region $\mathcal{X} = [0, \pi]^2$.

E.7.4 Hartmann

We select this function to see how the algorithms behave in *similar* functions as dimension increases. We do three versions of the Hartmann function, with dimensions $d = 3, 4,$ and 6. The equation is given by:

$$f(x) = \sum_{i=1}^{4}\alpha_i\exp\left(-\sum_{j=1}^{d}A_{ij}(x_j - P_{ij})\right)$$

where $\alpha = (1, 1.2, 3, 3.2)^T$. For $d = 3$ we use:

$$A = \begin{pmatrix} 3 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3 & 10 & 30 \\ 0.1 & 10 & 35 \end{pmatrix} \qquad P = 10^{-4}\begin{pmatrix} 3689 & 1170 & 2673 \\ 4699 & 4387 & 7470 \\ 1091 & 8732 & 5547 \\ 381 & 5743 & 8828 \end{pmatrix}$$

for $d = 4$ and $d = 6$ we use:

$$A = \begin{pmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix} \qquad P = 10^{-4}\begin{pmatrix} 1312 & 1696 & 5569 & 124 & 8283 & 5886 \\ 2329 & 4135 & 8307 & 3736 & 1004 & 9991 \\ 2348 & 1451 & 3522 & 2883 & 3047 & 6650 \\ 4047 & 8828 & 8732 & 5743 & 1091 & 381 \end{pmatrix}$$

They are all evaluated on the unit cube $[0, 1]^d$.

E.7.5 Perm10D

We select the 10-dimensional version of the Perm benchmark to test the capabilities of the algorithms in a very high-dimensional setting (by BO standards). The equation is given by:

$$f(x) = -10^{-21} \sum_{i=1}^{10} \left(\sum_{j=1}^{10} (j^i + \beta) \left(\left(\frac{x_j}{j} \right)^i - 1 \right) \right)^2$$

where we set $\beta = 10$. We evaluate it on $\mathcal{X} = [-10, 10]^d$.

E.7.6 SnAr4D

We implement the simulation using Summit [8]. We control temperature between 40 and 120 degrees, concentration from 0.1 to 0.5 moles per liter, and residence time between 0.5 and 2 minutes. We set the cost parameters for temperature $\alpha_{temp} = 5, \beta_{temp} = 1$, $\gamma_{temp} = 1$, for concentration $\alpha_{conc} = 2, \beta_{conc} = 0.01$, $\gamma_{conc} = 1$, and for residence time $\alpha_{residence} = 3$, $\beta_{residence} = 0.05$ and $\gamma_{residence} = 1$. We further optimize over the equivalents of pyrrolidine between 1 and 5 units, but we assume changing it incurs no input costs.

Since the SnAr benchmark is a multi-objective problem, we optimize a weighted sum of the two objectives:

$$\text{SnAr}(x) = \omega_1 \times \text{yield} - \omega_2 \times \text{e-factor}$$

where we set $\omega_1 = 10^{-4}$ and $\omega_2 = 0.1$. We optimize over $\mathcal{X} = [40, 120] \times [0.1, 0.5] \times [0.5, 2] \times [1, 5]$.

E.8 Ypacarai Implementation Details

As per [36] we model the Lake using the Shekel function which is given as:

$$f(x) = \sum_{i=1}^{m} \left(\sum_{j=1}^{2} (10x_j - C_{ji})^2 + \beta_i \right)^{-1}$$

For the objective we use $m = 2$, $m = 3$, and $m = 2$. The other parameters are:

$$C^{(1)} = \begin{pmatrix} 2 & 6.7 \\ 9 & 2 \end{pmatrix} \qquad C^{(2)} = \begin{pmatrix} 7 & 6 \\ 3.8 & 9.9 \\ 9 & 0.1 \end{pmatrix} \qquad C^{(3)} = \begin{pmatrix} 4 & 3 \\ 8.5 & 4 \end{pmatrix}$$

$$\beta^{(1)} = (9 \quad 9) \qquad \beta^{(2)} = (10 \quad 8 \quad 8) \qquad \beta^{(3)} = (7 \quad 9)$$

Where $C^{(i)}$ and $\beta^{(i)}$ represent the parameters of the ith objective. All objectives are optimized across on a subset of $\mathcal{X} = [0, 1]^2$. The subset is defined by creating grid of points mapping a high-resolution black and white image of Lake Ypacarai onto $\mathcal{X}$. For simplicity, we assume the cost of moving from one point to another is simply the distance between the points, even though in practice we might need to take a longer route to avoid land. For Truncated Expected Improvement, we simply project the truncation into the closest point grid, to avoid sampling points outside of the Lake.

F Full Experiment Results

F.1 Tables of Results

F.1.1 Synchronous Experiments

This section includes the full tables of results of the synthetic synchronous experiments. The results are shown in Table 3 and 4.

F.1.2 Asynchronous Experiments

In this section we include the full table results of all asynchronous experiments. The results are shown in Table 5 and 6.

Table 3: Comparison of 2-norm cost for different BO benchmark functions. The best three performances are shown in bold, and the best one in italics. We can see that SnAKe constantly achieves low cost, especially for larger budgets. The best cost performance is achieved by 0.0-SnAKe, however, we do this at the expense of worse regret. The only function for which SnAKe struggles is the very high dimensional Perm10D.

Method	Budget	0.0-SnAKe	0.1-SnAKe	1.0-SnAKe	ℓ-SnAKe	EI	EIpu	TrEI	UCB	PI	Random
Branin2D	15	4.5 ± 1.8	5.4 ± 1.6	5.8 ± 1.7	5.7 ± 1.5	6.7 ± 1.5	5.3 ± 1.2	5.1 ± 1.0	$\mathbf{4.4 \pm 1.6}$	$\mathbf{\mathit{1.0 \pm 0.9}}$	$\mathbf{4.1 \pm 0.4}$
	50	$\mathbf{5.8 \pm 3.0}$	9.3 ± 3.1	10 ± 4	9.8 ± 3.2	17 ± 6	$\mathbf{7.3 \pm 1.7}$	13.4 ± 2.3	15 ± 7	$\mathbf{\mathit{4.0 \pm 3.2}}$	7.5 ± 0.4
	100	$\mathbf{\mathit{5.4 \pm 2.9}}$	10 ± 4	10 ± 4	11 ± 4	37 ± 13	$\mathbf{9.1 \pm 1.7}$	25 ± 4	33 ± 17	12 ± 7	$\mathbf{10.2 \pm 0.5}$
	250	$\mathbf{\mathit{7.1 \pm 2.3}}$	15.4 ± 3.4	16 ± 4	$\mathbf{15.3 \pm 2.8}$	112 ± 32	$\mathbf{12.2 \pm 2.0}$	59 ± 13	$(9 \pm 5) \times 10^1$	32 ± 22	16.5 ± 0.7
Ackley4D	15	$\mathbf{14.7 \pm 1.2}$	$\mathbf{14.9 \pm 1.3}$	16.0 ± 1.0	16.4 ± 0.9	21.1 ± 0.8	15.4 ± 2.6	21.4 ± 0.8	19.5 ± 2.1	18.5 ± 3.4	$\mathbf{\mathit{8.0 \pm 0.4}}$
	50	$\mathbf{25 \pm 5}$	$\mathbf{28 \pm 4}$	31 ± 6	30 ± 5	66 ± 6	60 ± 8	69.4 ± 1.9	60 ± 9	52 ± 9	$\mathbf{\mathit{19.5 \pm 0.6}}$
	100	$\mathbf{\mathit{23 \pm 5}}$	27 ± 6	$\mathbf{24 \pm 6}$	$\mathbf{23 \pm 5}$	128 ± 9	123 ± 10	135.8 ± 3.2	107 ± 16	93 ± 10	32.6 ± 0.9
	250	$\mathbf{\mathit{25 \pm 8}}$	$\mathbf{32 \pm 9}$	$\mathbf{33 \pm 8}$	33 ± 6	302 ± 28	311 ± 17	330 ± 6	221 ± 30	210 ± 10	59.9 ± 1.1
Michaelwicz2D	15	$\mathbf{1.7 \pm 1.1}$	$\mathbf{1.9 \pm 0.6}$	1.9 ± 0.8	2.1 ± 1.1	11.8 ± 2.6	$\mathbf{\mathit{0.8 \pm 0.4}}$	2.3 ± 0.4	6.7 ± 1.6	4.2 ± 3.3	3.9 ± 0.4
	50	$\mathbf{2.1 \pm 0.8}$	3.0 ± 0.9	$\mathbf{2.7 \pm 0.8}$	3.1 ± 1.1	23 ± 4	$\mathbf{\mathit{1.58 \pm 0.33}}$	7.7 ± 0.9	24 ± 7	8 ± 12	7.5 ± 0.4
	100	$\mathbf{2.4 \pm 1.2}$	3.9 ± 0.8	$\mathbf{3.6 \pm 1.0}$	3.7 ± 1.0	30 ± 11	$\mathbf{\mathit{2.1 \pm 0.5}}$	15.6 ± 2.0	36 ± 4	17 ± 23	10.5 ± 0.4
	250	$\mathbf{\mathit{2.8 \pm 1.4}}$	5.8 ± 1.6	$\mathbf{4.7 \pm 1.6}$	5.4 ± 1.0	$(6 \pm 4) \times 10^1$	$\mathbf{3.1 \pm 0.4}$	40.3 ± 2.6	107 ± 22	$(6 \pm 5) \times 10^1$	16.4 ± 0.6
Hartmann3D	15	$\mathbf{\mathit{2.5 \pm 1.0}}$	3.4 ± 1.2	4.2 ± 1.3	3.6 ± 1.1	6.7 ± 3.0	$\mathbf{2.6 \pm 0.8}$	$\mathbf{3.1 \pm 0.6}$	5.9 ± 2.7	4.4 ± 3.1	6.24 ± 0.32
	50	$\mathbf{\mathit{4.3 \pm 2.1}}$	$\mathbf{5.7 \pm 2.2}$	6.9 ± 3.4	7.0 ± 3.3	15.1 ± 3.4	$\mathbf{4.9 \pm 2.6}$	5.8 ± 1.7	14 ± 4	9 ± 4	13.7 ± 0.5
	100	$\mathbf{\mathit{4.9 \pm 2.6}}$	9 ± 5	$\mathbf{8 \pm 4}$	9 ± 5	46 ± 8	$\mathbf{7.0 \pm 2.8}$	9.4 ± 2.8	26 ± 10	29 ± 12	21.5 ± 0.6
	250	$\mathbf{\mathit{4.9 \pm 2.3}}$	$\mathbf{10 \pm 4}$	$\mathbf{8.5 \pm 3.5}$	9.8 ± 3.4	96 ± 4	13.1 ± 3.5	20 ± 4	53 ± 31	$(9 \pm 4) \times 10^1$	38.5 ± 1.3
Hartmann6D	15	$\mathbf{\mathit{6 \pm 4}}$	$\mathbf{6 \pm 4}$	8 ± 4	8 ± 4	18 ± 4	$\mathbf{6.9 \pm 3.1}$	9 ± 4	17 ± 4	17 ± 5	10.5 ± 0.5
	50	$\mathbf{\mathit{11 \pm 5}}$	$\mathbf{11 \pm 4}$	12 ± 6	$\mathbf{12 \pm 4}$	61 ± 11	39 ± 9	32 ± 14	54 ± 9	49 ± 14	29.6 ± 0.8
	100	$\mathbf{\mathit{11 \pm 5}}$	$\mathbf{13 \pm 6}$	15 ± 9	$\mathbf{12 \pm 6}$	117 ± 21	94 ± 19	65 ± 29	102 ± 11	91 ± 28	51.8 ± 1.0
	250	$\mathbf{\mathit{13 \pm 6}}$	$\mathbf{15 \pm 7}$	15 ± 8	$\mathbf{15 \pm 9}$	$(2.7 \pm 0.6) \times 10^2$	$(2.5 \pm 0.7) \times 10^2$	$(1.6 \pm 0.7) \times 10^2$	224 ± 24	$(2.1 \pm 0.7) \times 10^2$	107.6 ± 1.3
Perm10D	15	22.2 ± 1.0	22.2 ± 1.1	22.3 ± 1.1	22.2 ± 1.2	23.7 ± 2.8	$\mathbf{6.3 \pm 0.5}$	$\mathbf{\mathit{2.0 \pm 0.6}}$	21.6 ± 2.8	$\mathbf{14 \pm 4}$	15.05 ± 0.27
	50	65.7 ± 2.6	65.7 ± 2.3	67.6 ± 2.3	66.0 ± 2.7	85 ± 11	$\mathbf{29.4 \pm 3.5}$	$\mathbf{\mathit{6.8 \pm 2.0}}$	76 ± 7	48 ± 14	$\mathbf{45.2 \pm 0.7}$
	100	118.5 ± 3.3	118.0 ± 3.2	129 ± 9	118 ± 4	173 ± 24	$\mathbf{67 \pm 7}$	$\mathbf{\mathit{14 \pm 4}}$	155 ± 12	98 ± 32	$\mathbf{82.5 \pm 0.9}$
	250	254 ± 12	254 ± 11	282 ± 11	251 ± 9	$(4.3 \pm 0.5) \times 10^2$	$\mathbf{202 \pm 24}$	$\mathbf{\mathit{35 \pm 10}}$	400 ± 32	$(2.6 \pm 0.8) \times 10^2$	$\mathbf{183.2 \pm 1.2}$

Table 4: Comparison of final $\log(\text{regret})$ for different BO benchmark functions. The best three performances are shown in bold, and the best one in italics. We can see that SnAKe constantly achieves regret comparable with classical Bayesian Optimization methods. The worst performance happens when $\epsilon = 0$, this could be explained by the method getting stuck in local optimums.

Method	Budget	0.0-SnAKe	0.1-SnAKe	1.0-SnAKe	ℓ-SnAKe	EI	EIpu	TrEI	UCB	PI	Random
Branin2D	15	-3.9 ± 1.7	-3.5 ± 0.7	-3.7 ± 1.6	-3.4 ± 0.9	$\mathbf{-4.7 \pm 1.5}$	$\mathbf{-4.5 \pm 1.4}$	-4.4 ± 1.3	$\mathbf{\mathit{-5.0 \pm 1.5}}$	-3.2 ± 1.5	-3.3 ± 0.8
	50	-6.2 ± 2.5	-8.1 ± 2.6	-7.9 ± 2.4	$\mathbf{-8.3 \pm 2.3}$	$\mathbf{\mathit{-8.7 \pm 1.7}}$	-7.0 ± 1.9	-6.1 ± 1.6	$\mathbf{-8.5 \pm 2.6}$	-6.2 ± 2.5	-4.4 ± 1.2
	100	-9.1 ± 3.2	$\mathbf{-11.2 \pm 2.2}$	$\mathbf{-11.4 \pm 2.7}$	-10.7 ± 2.2	$\mathbf{\mathit{-13 \pm 5}}$	-7.4 ± 2.0	-7.0 ± 2.0	-10.7 ± 2.5	-10.8 ± 1.7	-5.1 ± 1.1
	250	-12.3 ± 1.6	-13.2 ± 1.3	$\mathbf{-13.6 \pm 1.4}$	$\mathbf{-13.5 \pm 1.4}$	$\mathbf{\mathit{-15 \pm 7}}$	-7.9 ± 1.7	-8.1 ± 1.5	-11.9 ± 2.5	-12 ± 5	-6.3 ± 1.2
Ackley4D	15	1.62 ± 0.25	1.62 ± 0.25	$\mathbf{1.62 \pm 0.25}$	1.67 ± 0.18	1.71 ± 0.15	1.69 ± 0.20	1.73 ± 0.19	1.71 ± 0.15	$\mathbf{1.59 \pm 0.22}$	$\mathbf{\mathit{1.40 \pm 0.11}}$
	50	1.64 ± 0.19	1.64 ± 0.19	1.64 ± 0.19	1.65 ± 0.19	$\mathbf{1.59 \pm 0.22}$	1.69 ± 0.20	1.73 ± 0.19	1.70 ± 0.14	$\mathbf{1.34 \pm 0.21}$	$\mathbf{\mathit{1.21 \pm 0.14}}$
	100	1.70 ± 0.15	1.69 ± 0.15	1.69 ± 0.15	1.72 ± 0.11	$\mathbf{1.54 \pm 0.27}$	1.69 ± 0.20	1.73 ± 0.19	1.70 ± 0.14	$\mathbf{1.21 \pm 0.11}$	$\mathbf{\mathit{1.06 \pm 0.17}}$
	250	1.52 ± 0.22	$\mathbf{0.9 \pm 0.8}$	1.2 ± 0.5	1.0 ± 0.5	1.4 ± 0.5	1.69 ± 0.20	1.73 ± 0.19	1.5 ± 0.4	$\mathbf{\mathit{0.85 \pm 0.32}}$	$\mathbf{0.89 \pm 0.20}$
Michaelwicz2D	15	-4.2 ± 1.7	-4.8 ± 1.6	-5.1 ± 1.8	-4.5 ± 1.9	$\mathbf{-5.4 \pm 1.1}$	$\mathbf{-5.8 \pm 1.1}$	-5.4 ± 1.0	$\mathbf{\mathit{-5.9 \pm 1.3}}$	-4.3 ± 1.6	-4.5 ± 1.0
	50	-6.4 ± 1.5	$\mathbf{-7.3 \pm 1.4}$	-6.8 ± 1.3	-6.3 ± 1.4	-6.0 ± 1.3	-6.4 ± 1.1	$\mathbf{-7.0 \pm 1.1}$	$\mathbf{\mathit{-7.8 \pm 1.9}}$	-6.2 ± 1.7	-5.4 ± 0.4
	100	-7.0 ± 2.2	-7.2 ± 1.7	$\mathbf{-7.8 \pm 1.8}$	$\mathbf{-7.6 \pm 1.9}$	-6.2 ± 0.8	-6.6 ± 1.2	-7.5 ± 1.1	$\mathbf{\mathit{-8.2 \pm 2.0}}$	-6.6 ± 1.9	-6.0 ± 0.4
	250	-6.6 ± 1.6	$\mathbf{-8.1 \pm 2.2}$	$\mathbf{-8.5 \pm 2.2}$	-8.0 ± 2.2	-6.5 ± 0.7	-6.7 ± 1.3	-7.7 ± 1.0	$\mathbf{\mathit{-8.7 \pm 2.5}}$	-7.0 ± 1.4	-6.5 ± 0.7
Hartmann3D	15	-1.2 ± 1.5	-2.0 ± 1.9	-1.9 ± 1.7	-1.8 ± 1.6	$\mathbf{-2.2 \pm 1.6}$	$\mathbf{-2.7 \pm 1.5}$	$\mathbf{\mathit{-3.2 \pm 1.5}}$	-2.0 ± 1.7	-0.6 ± 1.1	-0.3 ± 0.6
	50	-2.9 ± 2.6	-4.8 ± 3.0	-4.8 ± 2.4	-4.8 ± 2.3	$\mathbf{\mathit{-7.4 \pm 1.6}}$	-5.0 ± 1.5	-5.1 ± 1.1	$\mathbf{-6.1 \pm 2.5}$	$\mathbf{-5.2 \pm 2.0}$	-1.4 ± 0.8
	100	-5.4 ± 2.9	$\mathbf{-8.3 \pm 1.5}$	-7.9 ± 2.4	-8.2 ± 2.1	$\mathbf{\mathit{-10.9 \pm 1.3}}$	-5.8 ± 1.5	-5.6 ± 1.0	-8.1 ± 3.2	$\mathbf{-9.9 \pm 2.6}$	-1.5 ± 0.5
	250	-6 ± 4	-9.8 ± 2.7	-9.2 ± 2.5	-9.4 ± 2.0	$\mathbf{\mathit{-12.4 \pm 2.0}}$	-6.8 ± 1.8	-6.7 ± 1.2	$\mathbf{-10 \pm 4}$	$\mathbf{-12.0 \pm 2.1}$	-2.4 ± 0.7
Hartmann6D	15	0.6 ± 0.5	$\mathbf{0.6 \pm 0.5}$	0.6 ± 0.5	$\mathbf{\mathit{0.5 \pm 0.6}}$	0.88 ± 0.33	0.8 ± 0.5	$\mathbf{0.5 \pm 0.5}$	0.77 ± 0.32	0.8 ± 0.4	0.7 ± 0.4
	50	0.3 ± 0.6	$\mathbf{0.1 \pm 0.7}$	$\mathbf{\mathit{-0.0 \pm 0.8}}$	$\mathbf{0.0 \pm 0.7}$	0.6 ± 0.4	0.3 ± 0.6	0.1 ± 0.6	0.4 ± 0.5	0.5 ± 0.6	0.49 ± 0.26
	100	$\mathbf{-0.2 \pm 0.8}$	$\mathbf{-0.3 \pm 1.0}$	-0.1 ± 0.9	$\mathbf{\mathit{-0.6 \pm 0.8}}$	0.2 ± 0.8	-0.1 ± 0.9	-0.1 ± 0.5	0.0 ± 0.6	0.0 ± 0.7	0.1 ± 0.5
	250	-0.6 ± 0.8	-0.7 ± 1.5	-0.7 ± 1.0	$\mathbf{-0.9 \pm 1.0}$	-0.7 ± 1.0	-0.5 ± 1.0	-0.4 ± 0.6	$\mathbf{\mathit{-0.9 \pm 0.8}}$	$\mathbf{-0.8 \pm 0.9}$	-0.04 ± 0.34
Perm10D	15	-2.0 ± 1.7	-2.0 ± 1.7	-2.1 ± 1.7	-2.3 ± 1.7	-5.4 ± 1.8	-4.9 ± 1.0	-4.3 ± 2.6	$\mathbf{\mathit{-6.6 \pm 1.6}}$	$\mathbf{-6.1 \pm 2.0}$	$\mathbf{-5.7 \pm 1.1}$
	50	-3.2 ± 1.9	-3.2 ± 1.9	-3.6 ± 2.0	-3.3 ± 1.8	-6.0 ± 2.2	$\mathbf{-7.5 \pm 1.9}$	-5.1 ± 2.5	$\mathbf{-7.5 \pm 1.7}$	-7.1 ± 1.9	$\mathbf{\mathit{-7.5 \pm 1.7}}$
	100	-3.4 ± 1.4	-3.4 ± 1.4	-4.7 ± 1.7	-3.7 ± 1.3	-6.4 ± 2.2	$\mathbf{-8.4 \pm 1.7}$	-6.4 ± 2.0	-8.0 ± 1.7	$\mathbf{\mathit{-8.5 \pm 1.8}}$	$\mathbf{-8.2 \pm 1.2}$
	250	-5.0 ± 2.3	-5.0 ± 2.3	-6.4 ± 2.3	-4.1 ± 2.1	-6.8 ± 2.4	$\mathbf{-9.3 \pm 1.2}$	-8.1 ± 1.0	-8.5 ± 1.9	$\mathbf{\mathit{-9.6 \pm 1.9}}$	$\mathbf{-9.5 \pm 1.6}$

F.1.3 SnAr Benchmark

We ran additional experiments on the SnAr benchmark. For the first one (which includes the example looked at in the main paper) we tested on a budget of $T = 100$ iterations for different values of t_{delay}. The results are included in Tables 7 and 8.

We also carried out synchronous results on the benchmark, for different budgets. The results are included in Tables 9 and 10.

F.2 Graphs for results of Section 4.1

We include the full graphs of the sequential Bayesian Optimization experiments. Each row represents a different budget. The left column shows the evolution of regret against the cost used, the middle column shows the evolution of regret with iterations, and the right column shows the evolution of the 2-norm cost. The results encompass Figures 11 to 16. The caption in each figure tells us the benchmark function being evaluated. Each experiment is the mean $\pm$ half the standard deviation of 10 different runs.

Table 5: Comparison of 2-norm cost for different benchmark functions in the *asynchronous setting*. The best three performances are shown in bold, and the best one in italics. SnAKe achieves considerable lower cost with respect to other methods, achieving the top 3 lowest costs each time.

Method	Budget	Delay	0.0-SnAKe	0.1-SnAKe	1.0-SnAKe	ℓ-SnAKe	Random	TS	UCBwLP	EIpuLP
Branin2D	100	10	***7.0 ± 1.9***	**10.0 ± 2.2**	10.6 ± 3.2	**9.8 ± 2.6**	10.4 ± 0.4	49 ± 5	27.5 ± 2.5	22 ± 5
	100	25	***7.8 ± 1.6***	11.2 ± 2.7	**9.6 ± 2.6**	10.6 ± 2.4	**10.2 ± 0.5**	52 ± 6	51 ± 4	25 ± 7
	250	10	***7.6 ± 1.7***	**14.5 ± 1.9**	15.9 ± 3.3	**14.9 ± 2.9**	16.6 ± 0.8	120 ± 12	37 ± 5	37 ± 14
	250	25	***8.8 ± 2.1***	**13.4 ± 1.5**	14.7 ± 3.3	**13.1 ± 2.7**	16.6 ± 0.6	122 ± 10	56 ± 4	48 ± 14
Ackley4D	100	10	***22.2 ± 2.3***	23.7 ± 1.9	**22.5 ± 2.0**	**22.4 ± 2.2**	32.5 ± 0.8	110 ± 6	96 ± 8	100 ± 26
	100	25	***19.6 ± 2.4***	24.0 ± 2.5	**22.9 ± 3.2**	**23.3 ± 3.5**	32.6 ± 0.9	101 ± 10	100 ± 4	59 ± 4
	250	10	**27 ± 4**	30.1 ± 2.9	**26.3 ± 2.6**	***26.2 ± 1.5***	59.6 ± 1.1	219 ± 23	243 ± 25	$(2.5 \pm 0.7) \times 10^2$
	250	25	**28.5 ± 3.3**	32 ± 4	***25.4 ± 2.3***	**25.6 ± 2.3**	59.5 ± 0.9	$(2.2 \pm 0.5) \times 10^2$	240 ± 20	175 ± 5
Michaelwicz2D	100	10	***3.4 ± 1.4***	5.0 ± 1.5	**4.5 ± 1.0**	**4.7 ± 1.0**	10.5 ± 0.4	17.8 ± 2.4	23.0 ± 2.6	54 ± 7
	100	25	***4.2 ± 1.1***	6.0 ± 1.2	**5.7 ± 1.3**	**5.9 ± 1.2**	10.5 ± 0.5	25.8 ± 2.6	33.5 ± 2.9	47.4 ± 3.3
	250	10	***3.5 ± 1.7***	**5.9 ± 0.8**	**5.7 ± 1.3**	6.4 ± 1.6	16.3 ± 0.9	27.2 ± 3.4	37 ± 7	67 ± 9
	250	25	***3.7 ± 1.1***	**6.6 ± 1.1**	6.7 ± 1.2	**6.6 ± 1.0**	16.4 ± 0.7	35.3 ± 2.6	36.0 ± 2.8	96 ± 9
Hartmann3D	100	10	***7.7 ± 3.4***	11 ± 4	**9.7 ± 3.2**	**10 ± 4**	21.7 ± 0.6	20.6 ± 3.3	34 ± 6	28 ± 4
	100	25	***9.6 ± 2.9***	**13 ± 4**	**12.8 ± 3.2**	14 ± 5	21.3 ± 0.5	32 ± 4	55 ± 5	49 ± 5
	250	10	***5.8 ± 2.4***	**11 ± 4**	**9.8 ± 3.1**	11 ± 4	38.6 ± 0.7	27 ± 4	42 ± 5	36 ± 5
	250	25	***7.6 ± 2.3***	**12 ± 4**	**11.2 ± 3.1**	12 ± 4	38.7 ± 0.9	38 ± 4	64 ± 9	57 ± 5
Hartmann4D	100	10	***12 ± 5***	**18 ± 6**	**18 ± 5**	21 ± 8	32.6 ± 0.7	39 ± 13	56 ± 9	47 ± 13
	100	25	***15 ± 4***	**20 ± 4**	**21 ± 5**	23 ± 5	32.5 ± 0.8	42 ± 8	71 ± 9	58 ± 7
	250	10	***13 ± 7***	**22 ± 12**	**20 ± 10**	22 ± 13	59.3 ± 1.0	$(8 \pm 5) \times 10^1$	103 ± 28	$(8 \pm 4) \times 10^1$
	250	25	***17 ± 11***	**28 ± 14**	**24 ± 11**	31 ± 15	59.4 ± 1.1	$(9 \pm 4) \times 10^1$	113 ± 20	96 ± 34
Hartmann6D	100	10	***14.9 ± 3.5***	**17 ± 5**	17.9 ± 3.5	**18 ± 4**	52.1 ± 1.2	39 ± 10	120 ± 19	124 ± 11
	100	25	***20.9 ± 2.7***	**22.4 ± 2.7**	23.9 ± 3.2	**24 ± 4**	51.7 ± 1.0	56 ± 10	109 ± 11	114 ± 9
	250	10	***18 ± 5***	**18 ± 5**	**18 ± 4**	20 ± 7	107.1 ± 1.6	53 ± 19	$(2.6 \pm 0.6) \times 10^2$	$(3.1 \pm 0.5) \times 10^2$
	250	25	***21 ± 5***	**23 ± 6**	24 ± 6	**23 ± 5**	107.0 ± 1.8	74 ± 18	$(2.7 \pm 0.5) \times 10^2$	306 ± 30

Table 6: Comparison of $\log(\text{regret})$ for different benchmark functions in the *asynchronous setting*. The best three performances are shown in bold, and the best one in italics. SnAKe achieves regret comparable with other Bayesian Optimization methods.

Method	Budget	Delay	0.0-SnAKe	0.1-SnAKe	1.0-SnAKe	ℓ-SnAKe	Random	TS	UCBwLP	EIpuLP
Branin2D	100	10	−9.6 ± 2.4	−9.7 ± 2.2	**−10.3 ± 2.4**	−10.1 ± 2.4	−5.7 ± 1.8	**−11.7 ± 1.1**	***−12.1 ± 1.5***	−7.3 ± 2.5
	100	25	−7.1 ± 2.9	**−8.3 ± 3.1**	−7.5 ± 2.6	−7.1 ± 2.2	−5.4 ± 1.0	***−11.7 ± 1.6***	**−8.2 ± 2.1**	−5.4 ± 1.7
	250	10	−11.9 ± 1.9	−12.9 ± 0.8	−12.8 ± 1.1	**−13.3 ± 1.2**	−5.9 ± 1.1	**−13.8 ± 1.0**	***−14.2 ± 1.6***	−9.3 ± 2.9
	250	25	−11.6 ± 2.0	−11.6 ± 0.8	**−12.1 ± 0.8**	−12.1 ± 1.1	−6.2 ± 1.1	***−15 ± 6***	**−14.8 ± 1.3**	−8.9 ± 2.7
Ackley4D	100	10	1.0 ± 0.5	**0.9 ± 0.7**	**0.9 ± 0.7**	***0.8 ± 0.7***	1.05 ± 0.18	1.46 ± 0.15	1.08 ± 0.21	1.4 ± 0.4
	100	25	1.21 ± 0.19	**1.1 ± 0.4**	1.15 ± 0.33	1.20 ± 0.23	**1.03 ± 0.17**	1.32 ± 0.16	***1.02 ± 0.23***	1.16 ± 0.17
	250	10	***−0.4 ± 0.8***	**−0.4 ± 1.0**	0.0 ± 0.8	**−0.2 ± 0.7**	0.8 ± 0.4	0.7 ± 0.8	0.1 ± 0.8	1.3 ± 0.4
	250	25	−0.3 ± 0.5	***−0.9 ± 0.5***	**−0.6 ± 0.5**	**−0.6 ± 0.5**	0.95 ± 0.13	0.4 ± 0.9	0.74 ± 0.29	1.02 ± 0.18
Michaelwicz2D	100	10	−7.1 ± 1.8	−7.1 ± 1.4	−6.8 ± 1.7	−7.4 ± 1.7	−6.2 ± 0.7	**−7.9 ± 2.5**	***−9.0 ± 1.1***	**−7.9 ± 1.5**
	100	25	−7.1 ± 2.1	**−7.4 ± 2.0**	−7.0 ± 1.4	−6.8 ± 1.1	−5.86 ± 0.21	−7.4 ± 1.6	***−11.2 ± 1.5***	**−9.2 ± 0.9**
	250	10	−6.8 ± 1.7	−8.3 ± 2.1	−8.5 ± 2.2	−8.4 ± 2.4	−6.5 ± 0.8	**−8.5 ± 3.2**	**−9.0 ± 1.1**	***−9.9 ± 1.7***
	250	25	−6.8 ± 1.3	−8.1 ± 2.0	**−8.6 ± 2.5**	−8.4 ± 2.3	−6.6 ± 0.9	−7.9 ± 2.4	***−11.2 ± 1.5***	**−10.3 ± 1.4**
Hartmann3D	100	10	−5.3 ± 2.8	**−7.8 ± 3.0**	−6.8 ± 3.4	−7.4 ± 2.7	−1.9 ± 0.9	**−9.5 ± 1.1**	***−10.0 ± 1.2***	−5.5 ± 1.3
	100	25	−4.7 ± 2.4	−6.0 ± 2.1	−6.1 ± 1.8	**−6.4 ± 1.7**	−1.6 ± 0.8	***−8.6 ± 1.3***	**−6.3 ± 1.5**	−4.3 ± 1.4
	250	10	−5 ± 4	−7.9 ± 3.5	−8.0 ± 3.5	**−8.6 ± 3.0**	−2.6 ± 0.8	**−10.7 ± 1.1**	***−12.8 ± 1.2***	−6.3 ± 1.3
	250	25	−6 ± 4	−8.4 ± 3.0	−8.8 ± 3.1	**−8.9 ± 2.6**	−2.4 ± 0.5	**−10.5 ± 0.7**	***−12.4 ± 0.9***	−6.2 ± 1.1
Hartmann4D	100	10	−3.1 ± 3.0	−2.9 ± 2.6	**−3.5 ± 2.4**	−2.8 ± 2.1	−1.1 ± 0.6	***−6.5 ± 2.8***	**−4.5 ± 1.9**	−2.6 ± 1.3
	100	25	**−2.1 ± 1.8**	−2.0 ± 1.7	−2.0 ± 1.9	−1.9 ± 1.3	−1.0 ± 0.5	***−4.2 ± 1.4***	−2.1 ± 1.0	**−2.2 ± 1.0**
	250	10	−6 ± 4	**−6 ± 4**	−6 ± 4	−4.2 ± 3.3	−1.2 ± 0.4	***−8.7 ± 2.1***	**−7.2 ± 1.2**	−4.6 ± 2.0
	250	25	−6 ± 4	−6 ± 4	**−6 ± 4**	−5.5 ± 3.2	−1.4 ± 0.5	***−8.7 ± 1.4***	**−6.5 ± 1.6**	−4.2 ± 1.7
Hartmann6D	100	10	**−0.2 ± 0.6**	***−0.3 ± 0.9***	−0.2 ± 0.7	**−0.3 ± 1.0**	0.1 ± 0.4	−0.2 ± 0.9	−0.0 ± 0.7	0.3 ± 0.5
	100	25	**−0.3 ± 0.7**	**−0.2 ± 0.5**	−0.2 ± 0.6	−0.2 ± 0.6	0.07 ± 0.29	***−0.4 ± 0.6***	0.0 ± 0.4	0.0 ± 0.5
	250	10	−0.5 ± 1.1	−0.7 ± 1.0	**−0.8 ± 0.7**	**−0.9 ± 1.6**	−0.2 ± 0.5	−0.5 ± 1.0	***−1.1 ± 0.7***	−0.2 ± 0.8
	250	25	−0.5 ± 0.8	−0.8 ± 0.9	−0.7 ± 1.1	***−1.1 ± 1.5***	−0.05 ± 0.30	**−0.9 ± 0.9**	**−1.0 ± 0.6**	−0.2 ± 0.5

F.3 Graphs for results of Section 4.1

We include the full graphs of the asynchronous Bayesian Optimization experiments. Each row represents a different budget. The left column shows the evolution of regret against the cost used, the middle column shows the evolution of regret with iterations, and the right columns show the evolution of the 2-norm cost. The results encompass Figures 17 to 28. The caption in each figure tells us the benchmark function being evaluated, and the time-delay for getting observations back. Each experiment is the mean $\pm$ half the standard deviation of 10 different runs.

F.4 Graphs for results of SnAr Benchmark (section 4.2)

Figure 29 includes the whole set of results of the SnAr benchmark in the asynchronous setting. Figure 30 includes the results for the synchronous setting. Each experiment is the mean $\pm$ half the standard deviation of 10 different runs.

Table 7: Comparison of cost on SnAr benchmark (asynchronous) for $T = 100$ and different values of t_{delay}. The best three performances are shown in bold, and the best one in italics. SnAKe consistently achieves lower cost than BO methods. EIpuLP achieves lower cost for small delays, suggesting over-exploration in the early stages, and under-exploration once observations arrive.

delay	0.0-SnAKe	0.1-SnAKe	1.0-SnAKe	ℓ-SnAKe	Random	TS	UCBwLP	EIpuLP
5	$(\mathbf{5.0 \pm 1.1}) \times \mathbf{10^2}$	$(5.7 \pm 0.9) \times 10^2$	$(\mathbf{5.5 \pm 0.8}) \times \mathbf{10^2}$	$(6.2 \pm 0.6) \times 10^2$	596 ± 26	$(1.12 \pm 0.05) \times 10^3$	$(9.2 \pm 0.6) \times 10^2$	$(\boldsymbol{1.6 \pm 0.5}) \times \boldsymbol{10^2}$
10	$(\mathbf{4.5 \pm 0.8}) \times \mathbf{10^2}$	$(\mathbf{5.3 \pm 0.7}) \times \mathbf{10^2}$	$(5.3 \pm 0.7) \times 10^2$	$(5.8 \pm 0.8) \times 10^2$	600 ± 20	$(1.11 \pm 0.06) \times 10^3$	$(8.9 \pm 0.8) \times 10^2$	$(\boldsymbol{1.9 \pm 0.4}) \times \boldsymbol{10^2}$
25	$(\mathbf{4.0 \pm 0.7}) \times \mathbf{10^2}$	$(4.8 \pm 0.6) \times 10^2$	$(\mathbf{4.5 \pm 0.8}) \times \mathbf{10^2}$	$(5.1 \pm 0.6) \times 10^2$	603 ± 25	$(1.09 \pm 0.06) \times 10^3$	$(9.3 \pm 0.9) \times 10^2$	$\boldsymbol{307 \pm 31}$
50	$(\boldsymbol{4.0 \pm 0.5}) \times \boldsymbol{10^2}$	$(\mathbf{4.5 \pm 0.4}) \times \mathbf{10^2}$	$(\mathbf{4.2 \pm 0.6}) \times \mathbf{10^2}$	$(4.6 \pm 0.5) \times 10^2$	606 ± 24	$(1.12 \pm 0.05) \times 10^3$	$(1.07 \pm 0.06) \times 10^3$	568 ± 27

Table 8: Comparison of regret on SnAr benchmark (asynchronous) for $T = 100$ and different values of t_{delay}. The best three performances are shown in bold, and the best one in italics. SnAKe consistently achieves regret comparable with BO methods. EIpuLP performs well for small delays, but poorly as the delay is increased.

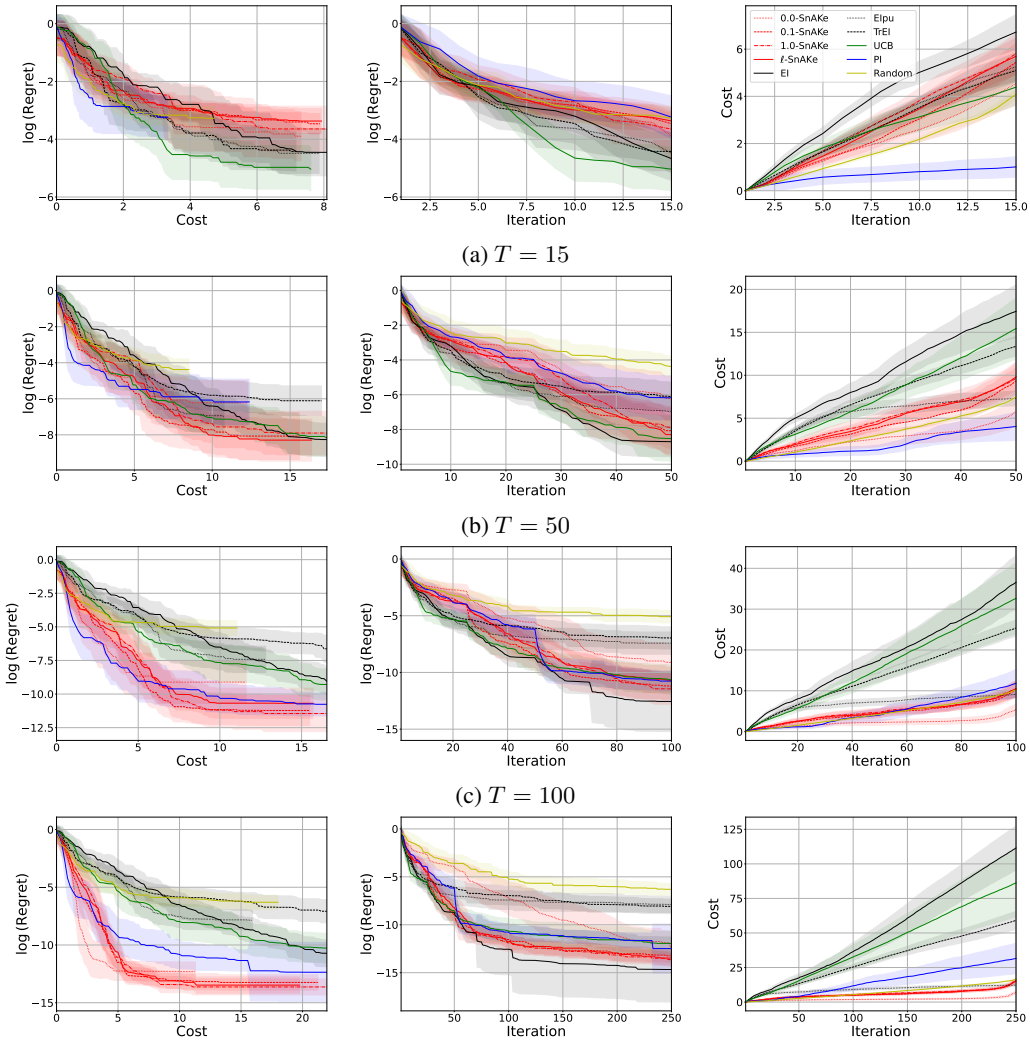
delay	0.0-SnAKe	0.1-SnAKe	1.0-SnAKe	ℓ-SnAKe	Random	TS	UCBwLP	EIpuLP
5	$\mathbf{-4.5 \pm 1.1}$	-4.5 ± 1.2	-4.3 ± 1.5	-3.5 ± 1.1	-0.93 ± 0.25	$\mathbf{-4.6 \pm 1.4}$	$\boldsymbol{-4.9 \pm 1.4}$	-4.1 ± 0.8
10	-3.5 ± 1.2	-3.9 ± 1.5	-3.7 ± 1.1	-4.0 ± 1.4	-1.1 ± 0.4	$\boldsymbol{-4.3 \pm 1.3}$	$\mathbf{-4.0 \pm 1.2}$	$\mathbf{-4.2 \pm 0.8}$
25	$\mathbf{-3.8 \pm 1.0}$	$\mathbf{-3.7 \pm 1.4}$	-3.3 ± 0.9	-3.6 ± 1.3	-0.9 ± 0.4	$\boldsymbol{-4.2 \pm 1.3}$	-2.9 ± 0.6	-3.1 ± 1.1
50	-3.2 ± 1.2	$\mathbf{-3.7 \pm 1.3}$	-3.1 ± 0.8	$\mathbf{-3.5 \pm 0.9}$	-1.1 ± 0.5	$\boldsymbol{-4.3 \pm 1.2}$	-3.2 ± 0.7	-2.3 ± 1.0

Table 9: Comparison of cost for SnAr benchmark (synchronous) for different budgets. The best three performances are shown in bold, and the best one in italics. SnAKe consistently achieves lower cost than BO methods. EIpu achieves the lowest cost in every instance, however Table 10 shows this is due to under-exploration.

Budget	0.0-SnAKe	0.1-SnAKe	1.0-SnAKe	ℓ-SnAKe	EI	EIpu	UCB	PI	Random
10	98 ± 13	101 ± 16	104 ± 12	103 ± 11	$\mathbf{89 \pm 11}$	$\boldsymbol{15 \pm 8}$	103 ± 10	$\mathbf{38 \pm 12}$	91 ± 7
25	219 ± 35	230 ± 25	247 ± 21	250 ± 21	230 ± 23	$\boldsymbol{46 \pm 24}$	289 ± 19	$\mathbf{108 \pm 24}$	$\mathbf{202 \pm 10}$
50	$(\mathbf{3.7 \pm 0.7}) \times \mathbf{10^2}$	$(4.0 \pm 0.5) \times 10^2$	$(4.3 \pm 0.6) \times 10^2$	$(4.1 \pm 0.7) \times 10^2$	503 ± 31	$(\boldsymbol{1.2 \pm 0.4}) \times \boldsymbol{10^2}$	$(5.5 \pm 0.4) \times 10^2$	$(3.9 \pm 0.4) \times 10^2$	$\mathbf{360 \pm 16}$
100	$(\mathbf{6.0 \pm 1.8}) \times \mathbf{10^2}$	$(6.9 \pm 1.3) \times 10^2$	$(7.0 \pm 1.5) \times 10^2$	$(7.2 \pm 1.3) \times 10^2$	$(1.12 \pm 0.04) \times 10^3$	$(\boldsymbol{3.0 \pm 0.9}) \times \boldsymbol{10^2}$	$(1.11 \pm 0.06) \times 10^3$	$(1.00 \pm 0.05) \times 10^3$	$\mathbf{605 \pm 26}$

Table 10: Comparison of $\log(\text{regret})$ for SnAr benchmark (synchronous) for different budgets. The best three performances are shown in bold, and the best one in italics. SnAKe achieves regret comparable with Bayesian Optimization methods. EIpu achieves the worst non-random performance in every instance.

Budget	0.0-SnAKe	0.1-SnAKe	1.0-SnAKe	ℓ-SnAKe	EI	EIpu	UCB	PI	Random
10	-1.9 ± 0.9	-2.0 ± 0.9	$\mathbf{-2.2 \pm 1.1}$	-1.6 ± 0.8	$\boldsymbol{-2.2 \pm 1.0}$	-1.0 ± 0.8	$\mathbf{-2.0 \pm 1.0}$	-1.0 ± 1.0	-0.29 ± 0.26
25	$\mathbf{-3.4 \pm 0.8}$	-3.3 ± 0.7	-3.3 ± 1.1	-2.9 ± 0.8	$\mathbf{-3.6 \pm 0.7}$	-2.9 ± 0.9	-3.0 ± 0.8	$\boldsymbol{-3.8 \pm 1.3}$	-0.51 ± 0.27
50	-4.5 ± 1.1	$\mathbf{-4.5 \pm 1.0}$	-4.1 ± 1.2	-4.3 ± 1.0	$\boldsymbol{-4.9 \pm 0.8}$	-3.7 ± 1.0	-4.2 ± 0.9	$\mathbf{-4.9 \pm 0.7}$	-0.9 ± 0.5
100	-5.3 ± 1.3	-5.8 ± 1.1	$\mathbf{-5.9 \pm 1.2}$	-5.6 ± 1.0	$\mathbf{-6.0 \pm 0.8}$	-4.9 ± 1.0	$\boldsymbol{-6.0 \pm 0.5}$	-5.8 ± 0.7	-1.0 ± 0.4



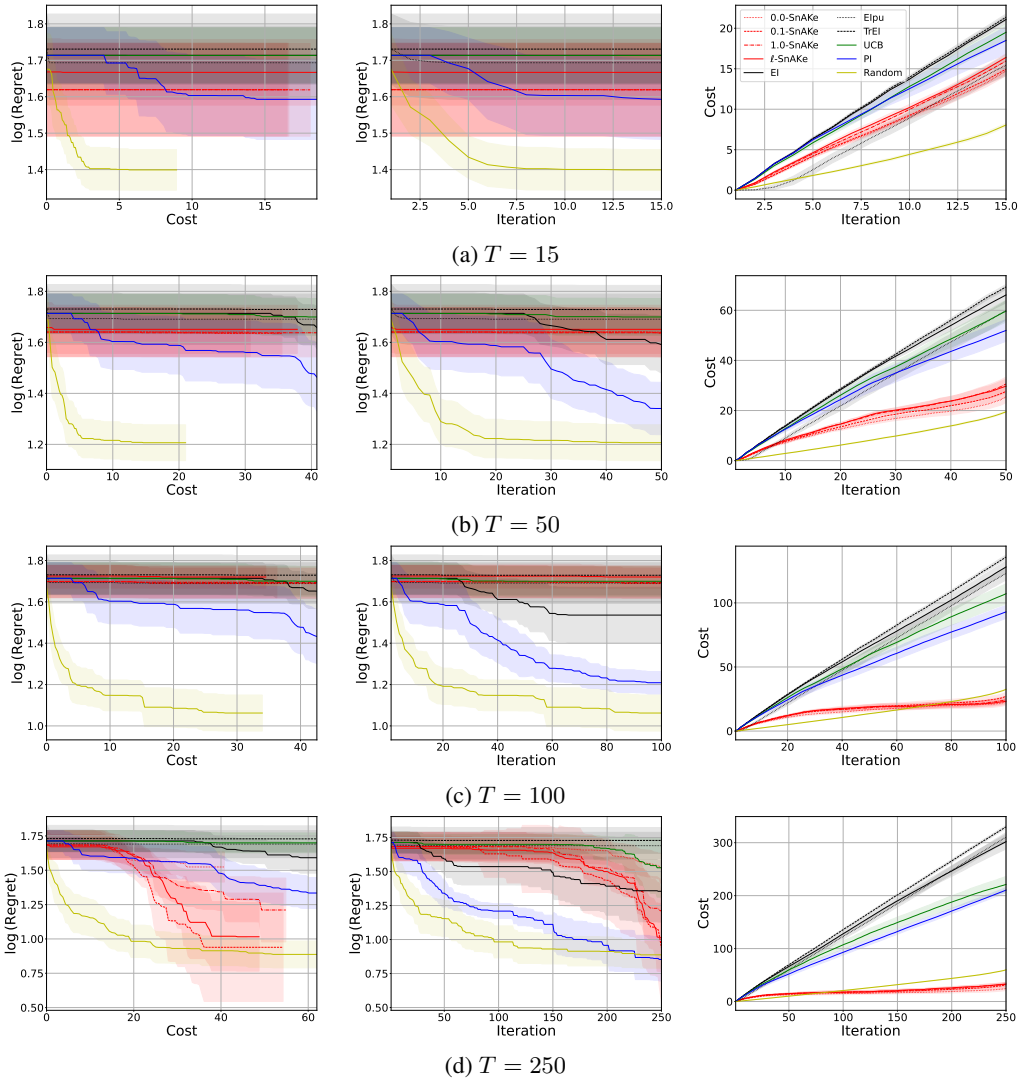
(a) $T = 15$

(b) $T = 50$

(c) $T = 100$

(d) $T = 250$

Figure 11: Branin2D. Each row represents a different budget. The left column shows the evolution of regret against the cost used. The middle column shows the evolution of regret with iterations, and the right columns show the evolution of the 2-norm cost. As we increase the budget, SnAKe outperforms two BO methods in regret, and outperforms all methods in cost. $\epsilon = 0$ gives the smallest cost of all at the expense of some regret.



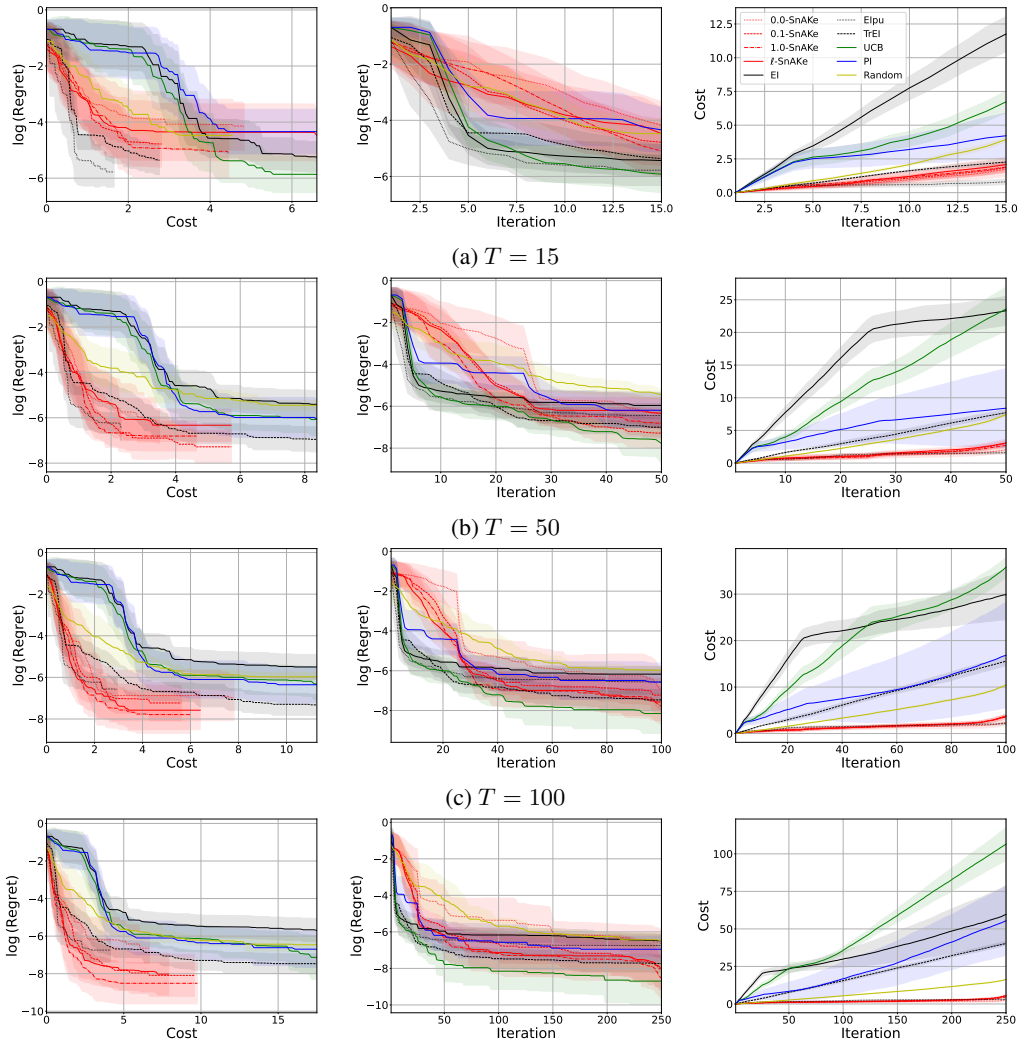
(a) $T = 15$

(b) $T = 50$

(c) $T = 100$

(d) $T = 250$

Figure 12: Ackley4D. Each row represents a different budget. The left column shows the evolution of regret against the cost used. The middle column shows the evolution of regret with iterations, and the right columns show the evolution of the 2-norm cost. SnAKe performs badly for smaller budgets, this may be because of the Thompson Sampling (see Figure 20, TS performs very badly in asynchronous Ackely). For the largest budget SnAKe recovers and outperforms all other methods in *both* regret and cost.



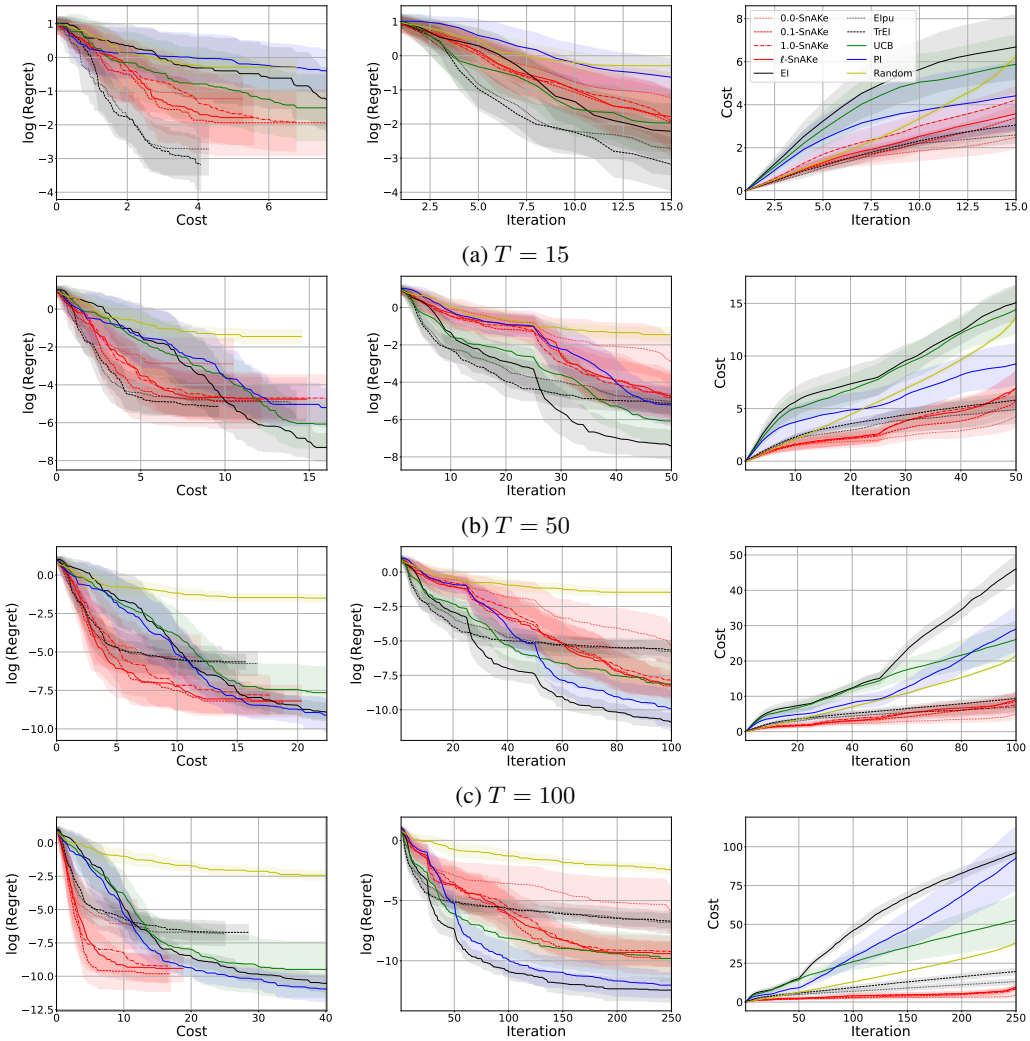
(a) $T = 15$

(b) $T = 50$

(c) $T = 100$

(d) $T = 250$

Figure 13: Michaelwicz2D. Each row represents a different budget. The left column shows the evolution of regret against the cost used. The middle column shows the evolution of regret with iterations, and the right columns show the evolution of the 2-norm cost. SnAKe outperforms has regret comparable with other methods for all budgets (UCB outperforms the rest for larger ones). SnAKe achieves significantly less cost at all budgets, this may be due to SnAKe exploring the many local optimums carefully. The first column shows that SnAKe achieves by far the best regret for low cost. In this example, SnAKe and EIpu have similar performance.



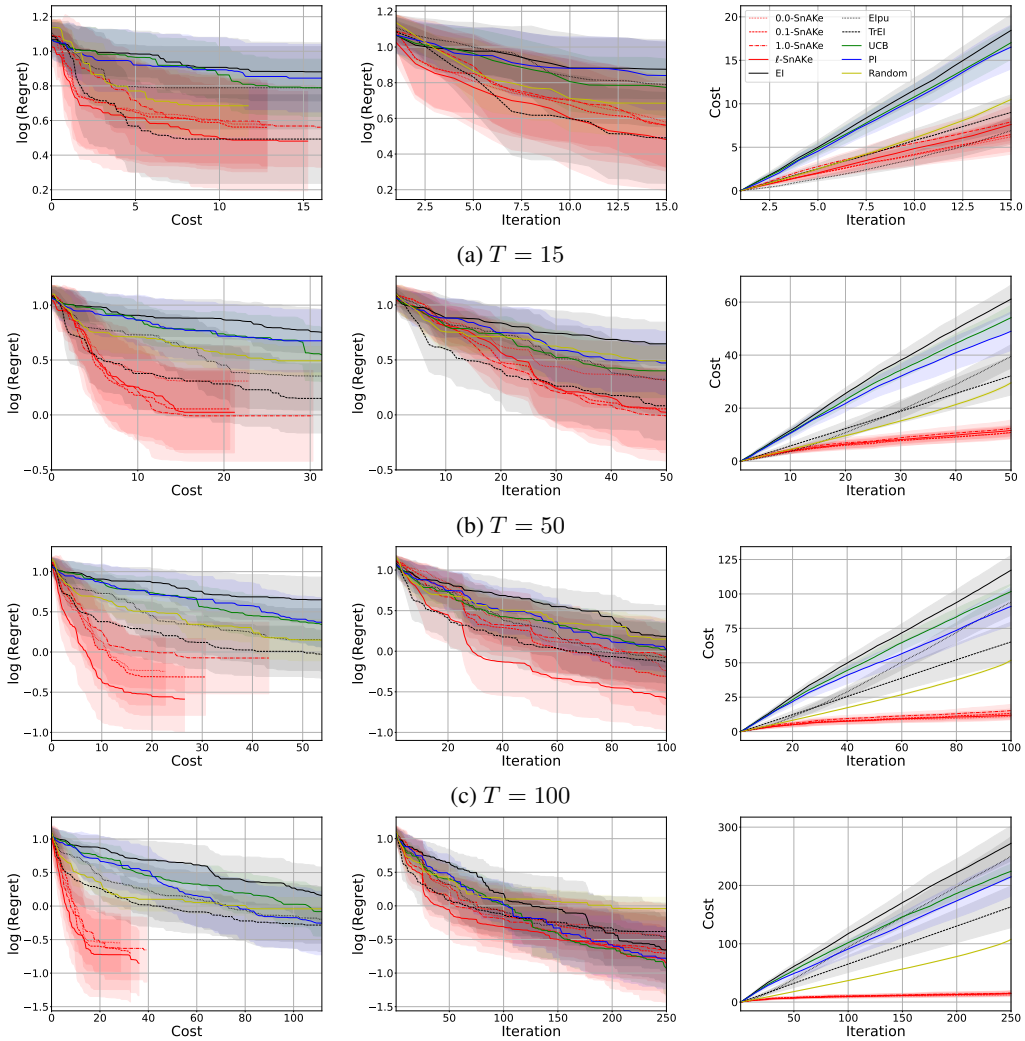
(a) $T = 15$

(b) $T = 50$

(c) $T = 100$

(d) $T = 250$

Figure 14: Hartmann3D. Each row represents a different budget. The left column shows the evolution of regret against the cost used. The middle column shows the evolution of regret with iterations, and the right columns show the evolution of the 2-norm cost. Again, SnAKe achieves the best regret at low cost for all budgets. $\epsilon = 0$ struggles in this benchmark, showcasing the impact that Point Deletion can have. EIpu and TrEI achieve higher cost and worse regret than SnAKe.



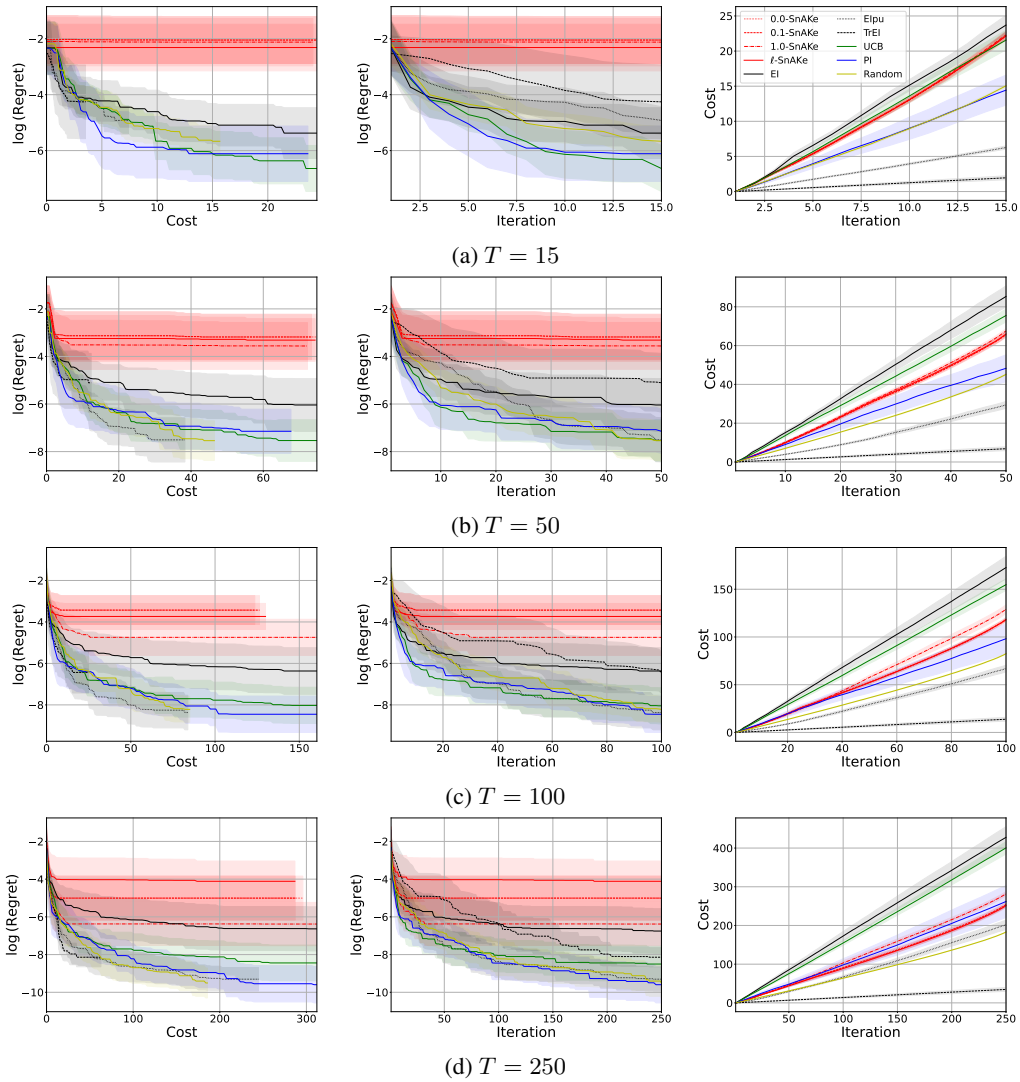
(a) $T = 15$

(b) $T = 50$

(c) $T = 100$

(d) $T = 250$

Figure 15: Hartmann6D. Each row represents a different budget. The left column shows the evolution of regret against the cost used. The middle column shows the evolution of regret with iterations, and the right columns show the evolution of the 2-norm cost. A high-dimensional example where SnAKe performs exceedingly well, giving the best regret at low costs for all budgets except $T = 15$. The final cost is considerably lower for SnAKe than any other method.



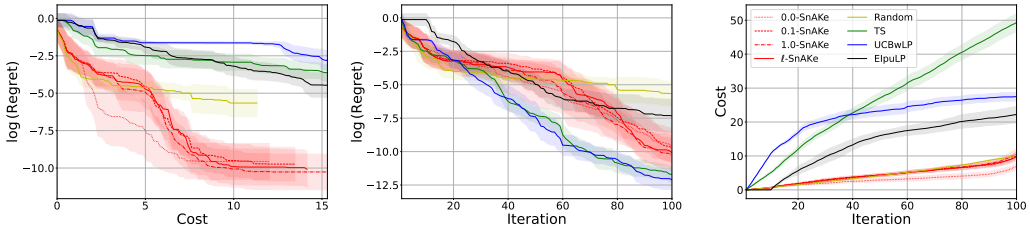
(a) $T = 15$

(b) $T = 50$

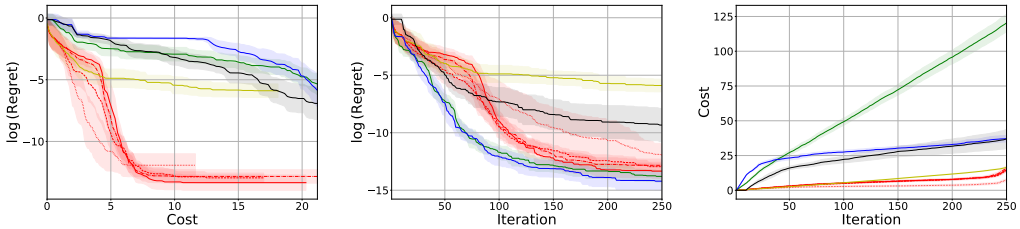
(c) $T = 100$

(d) $T = 250$

Figure 16: Perm10D. Each row represents a different budget. The left column shows the evolution of regret against the cost used. The middle column shows the evolution of regret with iterations, and the right columns show the evolution of the 2-norm cost. SnAKe struggles in this benchmark, however, EI also struggles. As an interesting observation, if we *did not* update the model, we would achieve a much better performance (as it would be equivalent to Random). We observe this behavior in the asynchronous case, where having a time-delay helps the method perform better (see asynchronous Ackley, Figure 20). EIpu and TrEI perform well in this example, we conjecture this is because they are doing far more localized searches while SnAKe is trying to cover all the space available (which is very difficult in higher dimensions).

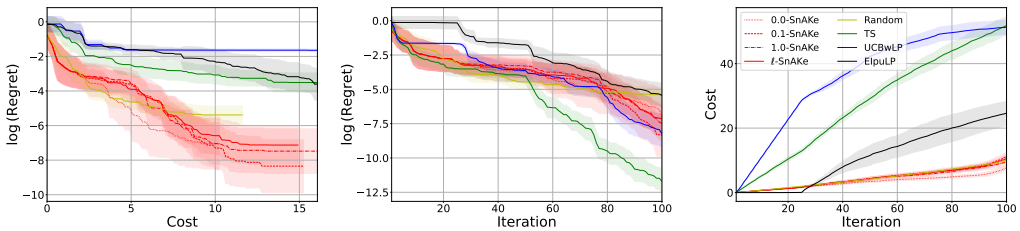


(a) $T = 100$

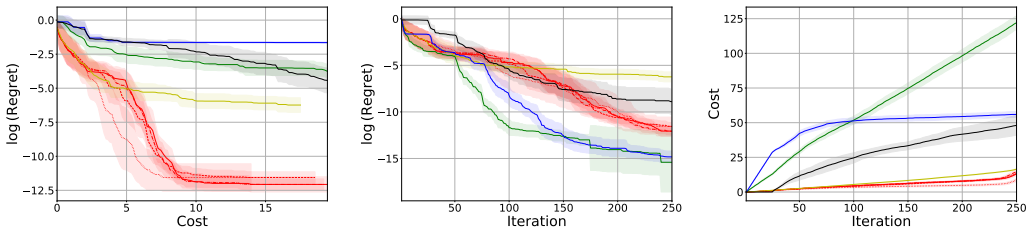


(b) $T = 250$

Figure 17: Branin2D (Asynchronous), $t_{delay} = 10$. Each row represents a different budget. The left column shows the evolution of regret against the cost used. The middle column shows the evolution of regret with iterations, and the right columns show the evolution of the 2-norm cost. SnAKe achieves significantly better regret than all other methods at low costs. The final regret of other BO methods is slightly better, but this comes at the expense of much larger cost. EIpuLP performs poorly, as it seems Local Penalization is over-powering the cost term.

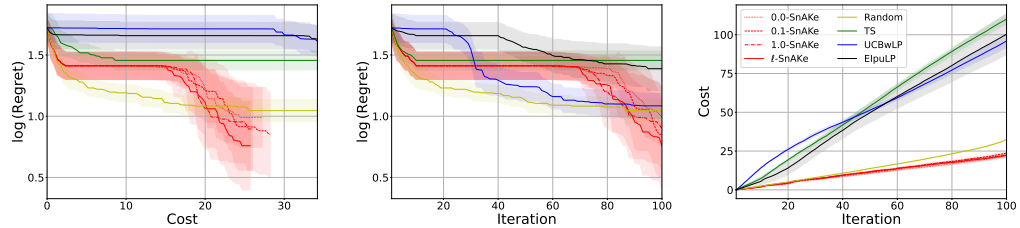


(a) $T = 100$

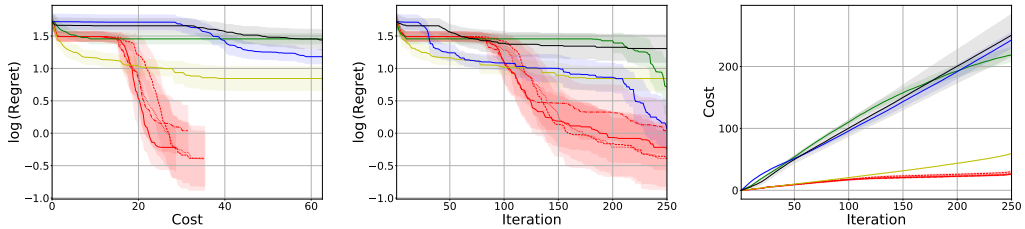


(b) $T = 250$

Figure 18: Branin2D (Asynchronous), $t_{delay} = 25$. Each row represents a different budget. The left column shows the evolution of regret against the cost used. The middle column shows the evolution of regret with iterations, and the right columns show the evolution of the 2-norm cost. The results are similar to the shorter delay seen in Figure 17.

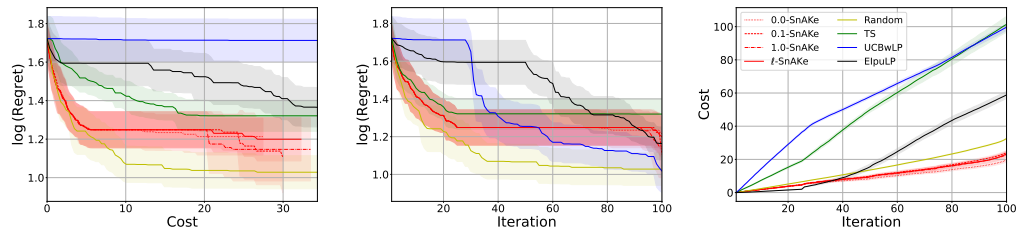


(a) $T = 100$

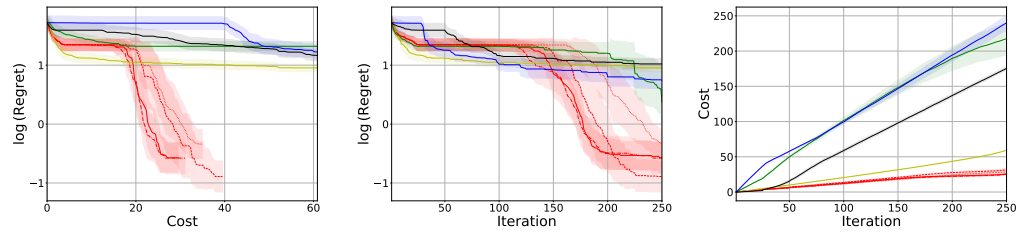


(b) $T = 250$

Figure 19: Ackley4D (Asynchronous), $t_{delay} = 10$. Each row represents a different budget. The left column shows the evolution of regret against the cost used. The middle column shows the evolution of regret with iterations, and the right columns show the evolution of the 2-norm cost. For the larger budget, SnAKe outperforms all other methods in both regret and cost. Interestingly, the performance of SnAKe *improves* when adding delay (see Figure 12 for synchronous results). EIpuLP performs poorly, as it seems Local Penalization is over-powering the cost term.

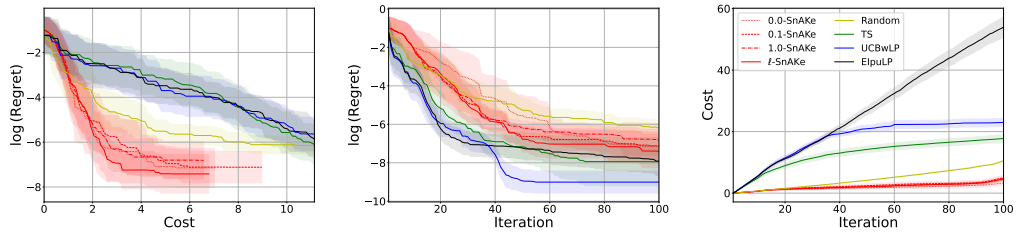


(a) $T = 100$

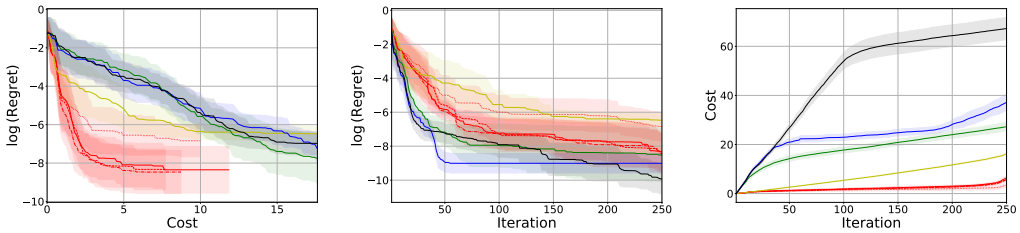


(b) $T = 250$

Figure 20: Ackley4D (Asynchronous), $t_{delay} = 25$. Each row represents a different budget. The left column shows the evolution of regret against the cost used. The middle column shows the evolution of regret with iterations, and the right columns show the evolution of the 2-norm cost. Results are similar to the case when $t_{delay} = 10$, see Figure 19.

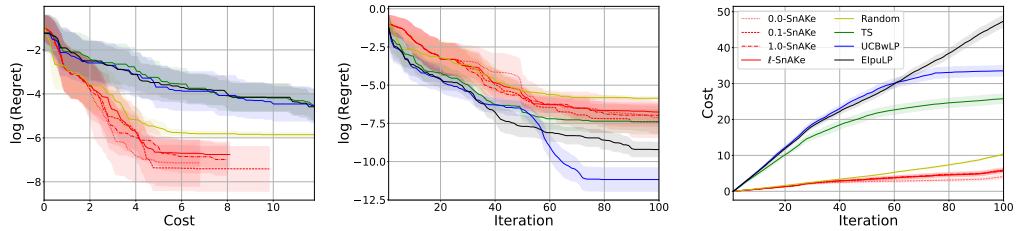


(a) $T = 100$

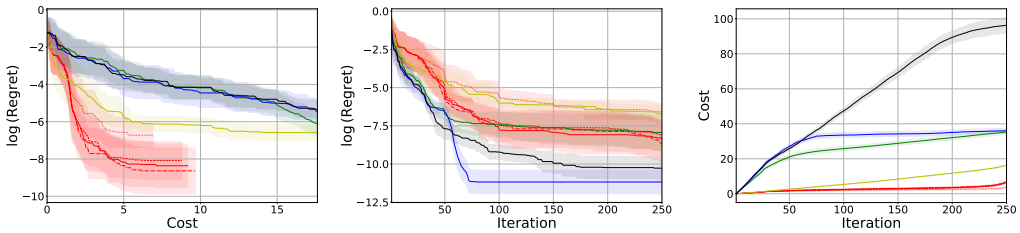


(b) $T = 250$

Figure 21: Michaelwicz2D (Asynchronous), $t_{delay} = 10$. Each row represents a different budget. The left column shows the evolution of regret against the cost used. The middle column shows the evolution of regret with iterations, and the right columns show the evolution of the 2-norm cost. For this benchmark, UCBwLP achieves the best regret by far, however, at the cost of significant expense. For low cost, SnAKe achieves much better regret.

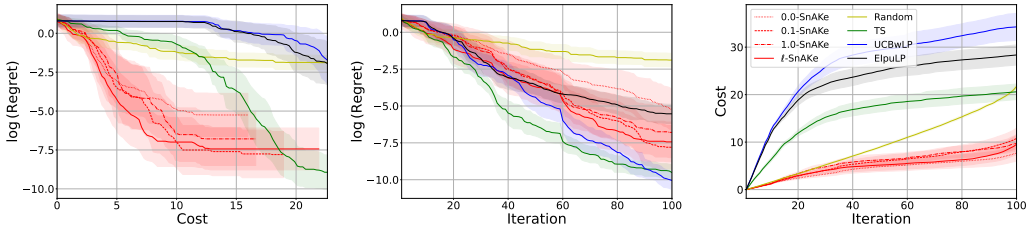


(a) $T = 100$

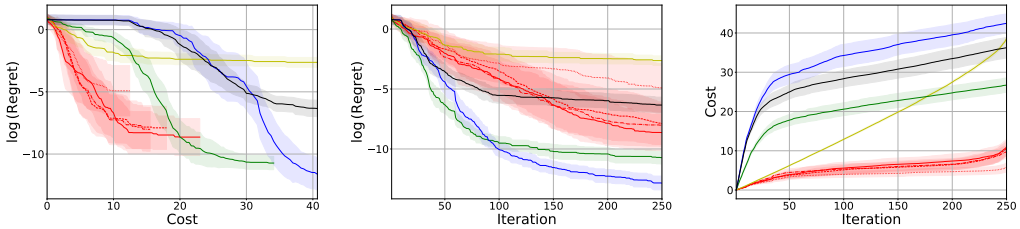


(b) $T = 250$

Figure 22: Michaelwicz2D (Asynchronous), $t_{delay} = 25$. Each row represents a different budget. The left column shows the evolution of regret against the cost used. The middle column shows the evolution of regret with iterations, and the right columns show the evolution of the 2-norm cost. Similar results to shorter delay, see Figure 21.

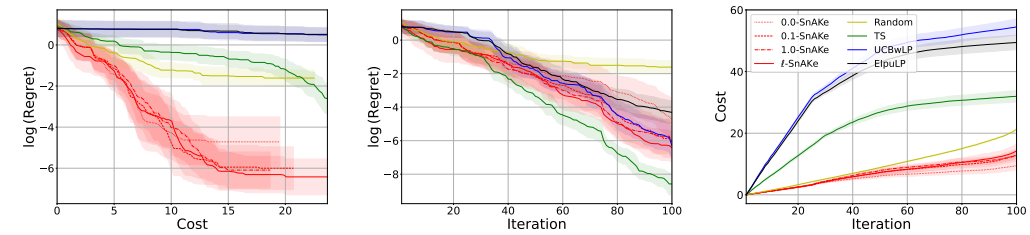


(a) $T = 100$

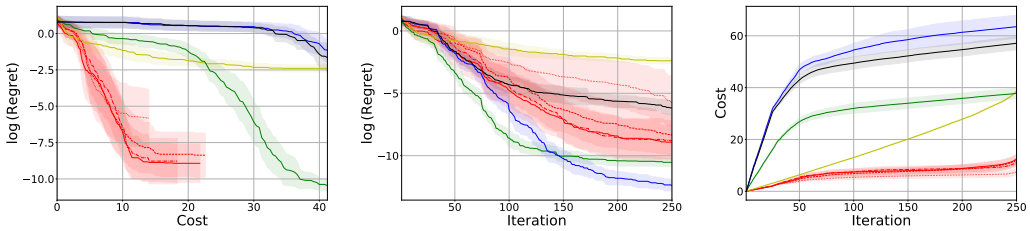


(b) $T = 250$

Figure 23: Hartmann3D (Asynchronous), $t_{delay} = 10$. Each row represents a different budget. The left column shows the evolution of regret against the cost used. The middle column shows the evolution of regret with iterations, and the right columns show the evolution of the 2-norm cost. SnAKe achieves the best regret for low cost, with Thompson Sampling also giving a good performance. For the full optimization, UCBwLP achieves the best regret, at the expense of four times the cost of SnAKe. EIpuLP performs poorly, as it seems Local Penalization is over-powering the cost term.

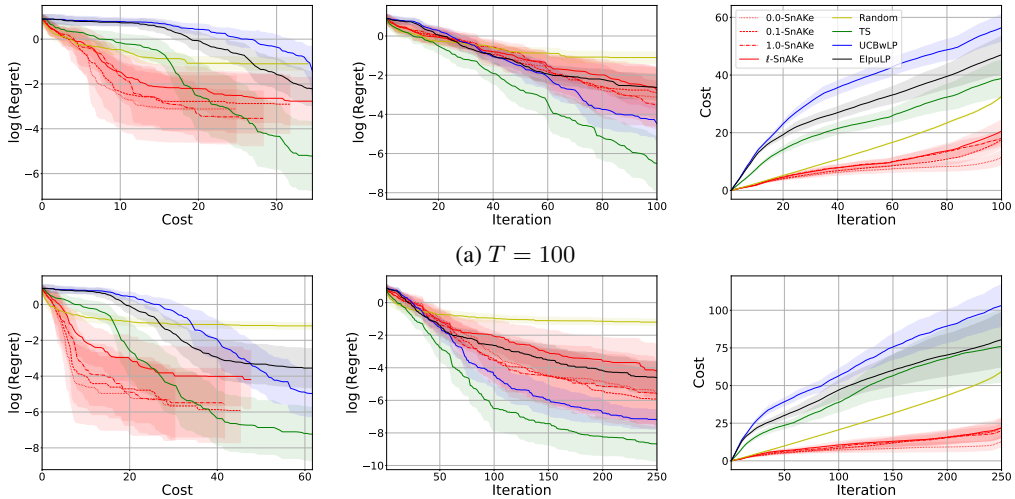


(a) $T = 100$



(b) $T = 250$

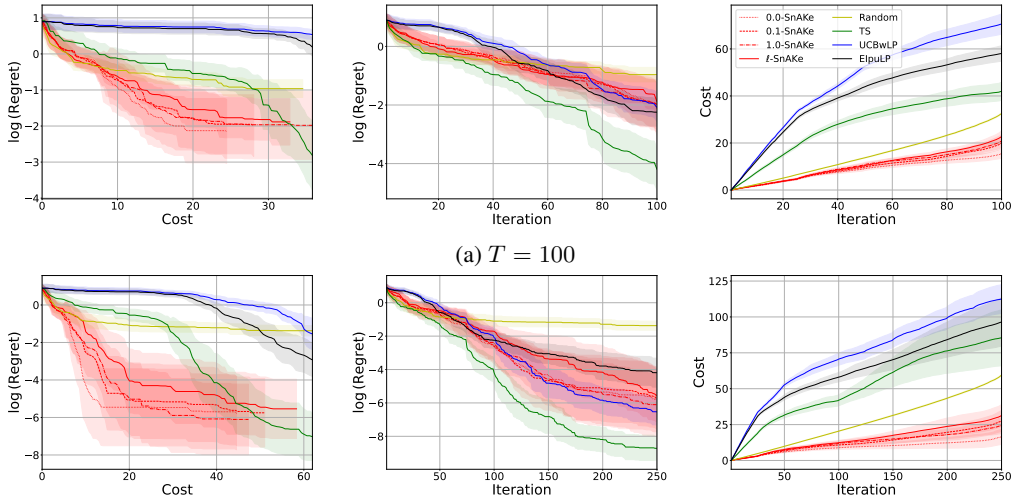
Figure 24: Hartmann3D (Asynchronous), $t_{delay} = 25$. Each row represents a different budget. The left column shows the evolution of regret against the cost used. The middle column shows the evolution of regret with iterations, and the right columns show the evolution of the 2-norm cost. Similar results to the case with smaller delay, see Figure 23.



(a) $T = 100$

(b) $T = 250$

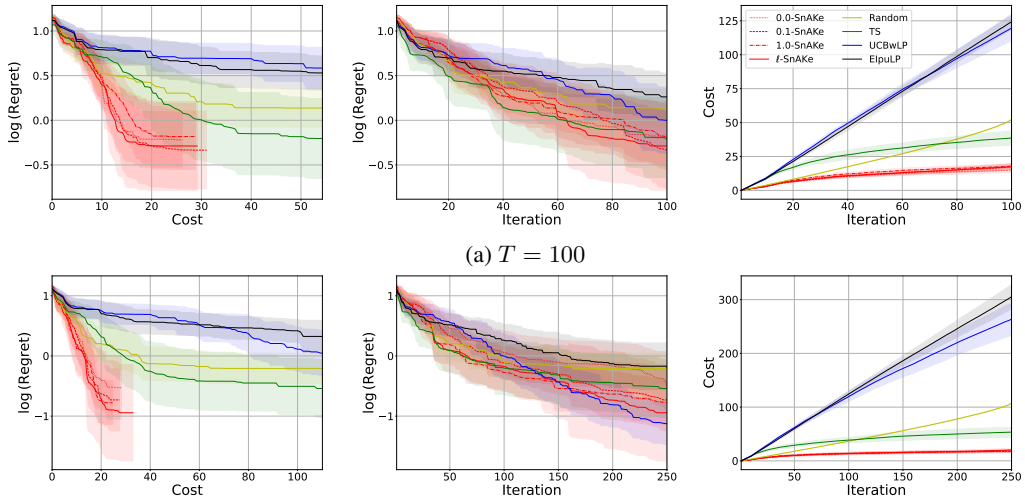
Figure 25: Hartmann4D (Asynchronous), $t_{delay} = 10$. Each row represents a different budget. The left column shows the evolution of regret against the cost used. The middle column shows the evolution of regret with iterations, and the right columns show the evolution of the 2-norm cost. Similar results to other Hartmann benchmarks, see Figure 23.



(a) $T = 100$

(b) $T = 250$

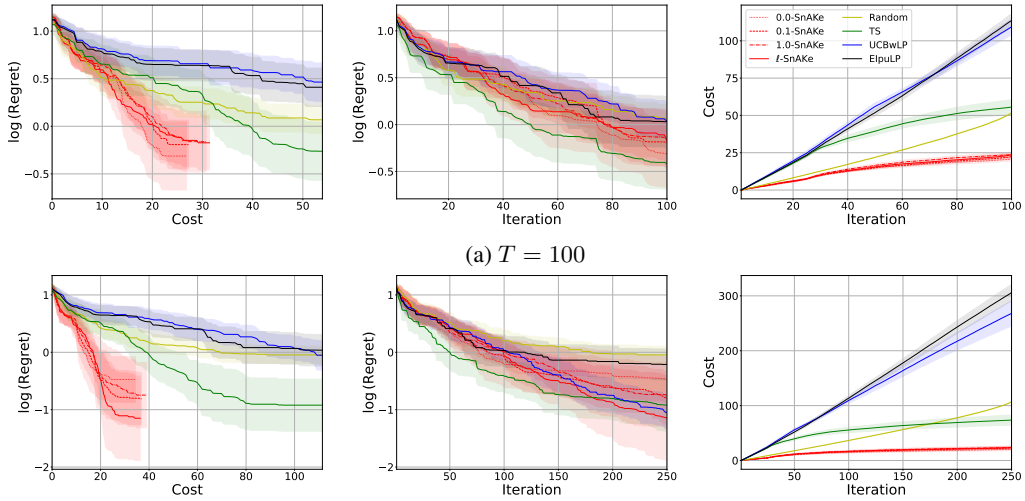
Figure 26: Hartmann4D (Asynchronous), $t_{delay} = 25$. Each row represents a different budget. The left column shows the evolution of regret against the cost used. The middle column shows the evolution of regret with iterations, and the right columns show the evolution of the 2-norm cost. Similar results to other Hartmann benchmarks, see Figure 23.



(a) $T = 100$

(b) $T = 250$

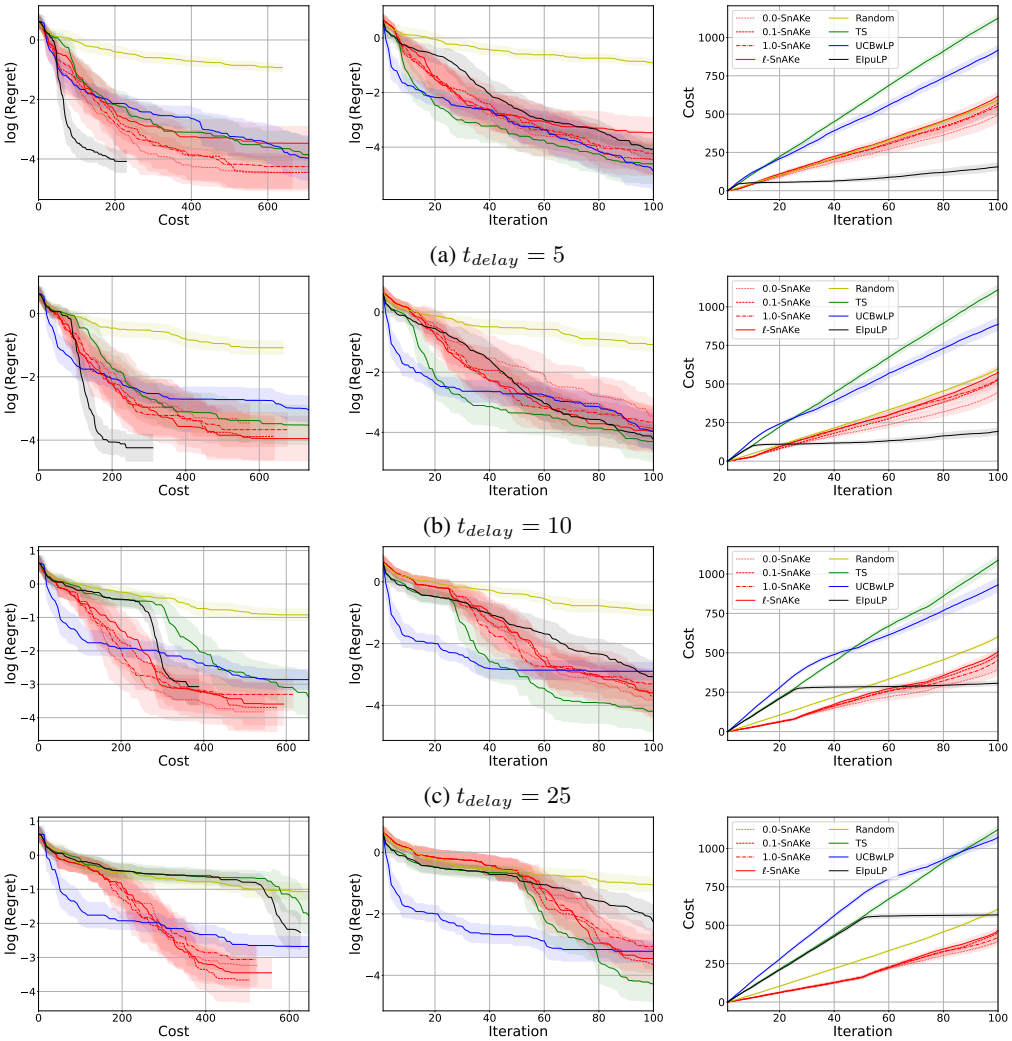
Figure 27: Hartmann6D (Asynchronous), $t_{delay} = 10$. Each row represents a different budget. The left column shows the evolution of regret against the cost used. The middle column shows the evolution of regret with iterations, and the right columns show the evolution of the 2-norm cost. Similar results to other Hartmann benchmarks, see Figure 23.



(a) $T = 100$

(b) $T = 250$

Figure 28: Hartmann6D (Asynchronous), $t_{delay} = 25$. Each row represents a different budget. The left column shows the evolution of regret against the cost used. The middle column shows the evolution of regret with iterations, and the right columns show the evolution of the 2-norm cost. Similar results to other Hartmann benchmarks, see Figure 23.



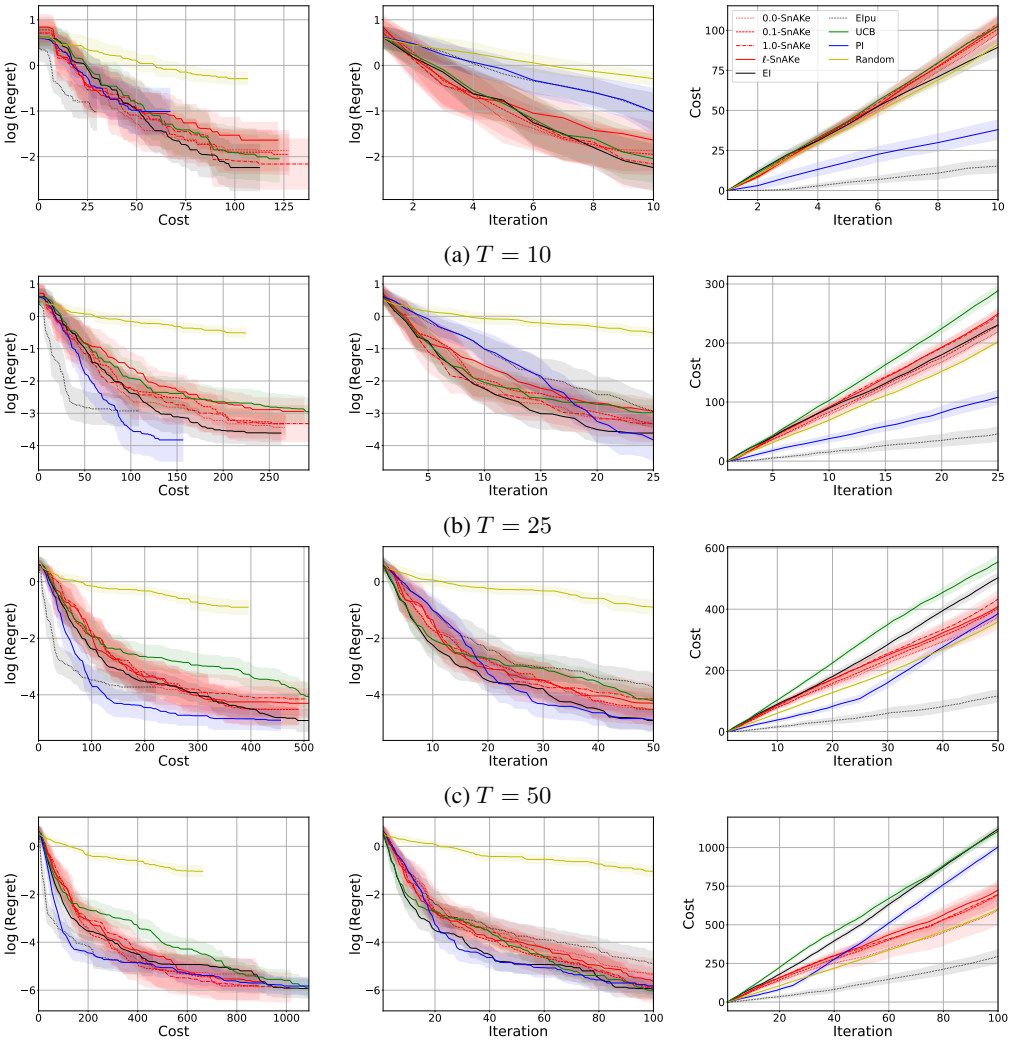
(a) $t_{delay} = 5$

(b) $t_{delay} = 10$

(c) $t_{delay} = 25$

(d) $t_{delay} = 50$

Figure 29: SnAr benchmark (Asynchronous) with $T = 100$. Each row represents a different t_{delay}. The left column shows the evolution of regret against the cost used. The middle column shows the evolution of regret with iterations, and the right columns show the evolution of the cost as defined in Section 4.2. SnAKe achieves the better regret than classical BO algorithms at low cost for all budgets. EIpuLP performs well for small delays, but poorly for larger delays.



(a) $T = 10$

(b) $T = 25$

(c) $T = 50$

(d) $T = 100$

Figure 30: SnAr benchmark (synchronous, $t_{delay} = 1$). Each row shows a different budget. The left column shows the evolution of regret against the cost used. The middle column shows the evolution of regret with iterations, and the right columns show the evolution of the cost as defined in Section 4.2. SnAKe is the only method achieving low regret and low cost especially for larger budgets. EIpu generally achieves low cost but poor regret.