Adversarial Attacks on Gaussian Process Bandits

Eric Han [1] **Jonathan Scarlett** [1 2]

Abstract

Gaussian processes (GP) are a widely-adopted tool used to sequentially optimize black-box functions, where evaluations are costly and potentially noisy. Recent works on GP bandits have proposed to move beyond random noise and devise algorithms robust to *adversarial attacks*. This paper studies this problem from the attacker's perspective, proposing various adversarial attack methods with differing assumptions on the attacker's strength and prior information. Our goal is to understand adversarial attacks on GP bandits from theoretical and practical perspectives. We focus primarily on *targeted* attacks on the popular GP-UCB algorithm and a related elimination-based algorithm, based on adversarially perturbing the function f to produce another function $\tilde{f}$ whose optima are in some target region $\mathcal{R}_{\text{target}}$. Based on our theoretical analysis, we devise both white-box attacks (known f) and black-box attacks (unknown f), with the former including a Subtraction attack and Clipping attack, and the latter including an Aggressive subtraction attack. We demonstrate that adversarial attacks on GP bandits can succeed in forcing the algorithm towards $\mathcal{R}_{\text{target}}$ even with a low attack budget, and we test our attacks' effectiveness on a diverse range of objective functions.

1. Introduction

Gaussian Processes (GPs) are commonly used to sequentially optimize unknown objective functions whose evaluations are costly. This method has successfully been applied to a litany of applications, such as hyperparameter tuning (Snoek et al., 2012; Swersky et al., 2013), robotics (Jaquier et al., 2020), recommender systems (Vanchinathan et al., 2014) and more. In many of these applications, corruptions in the measurements are not sufficiently well captured by random noise alone. For instance, one may be faced with rare outliers (e.g., due to equipment failures), or bad actors may influence the observations (e.g., malicious users in recommender systems).

These uncertainties have been addressed via the consideration of an *adversary* in the GP bandit optimization problem; function observations are not only subject to random noise, but also adversarial noise. With this additional requirement, the optimization not only becomes more robust to uncertainty, but can also maintain robustness in the presence of malicious adversaries. The notion of adversarial attacks on bandit algorithms appears to have been inspired by the extensive literature on adversarial attacks on deep neural networks (Szegedy et al., 2014), although the associated approaches are generally different.

1.1. Related Work

A myriad of GP optimization methods have been developed in the literature to overcome the various forms of uncertainty, and achieve some associated notion of robustness:

- *Presence of outliers*, where some function evaluations are highly unreliable (Martinez-Cantin et al., 2018).
- *Random perturbations to sampled points*, where the sampled points are subject to random uncertainty (Beland & Nair, 2017; Nogueira et al., 2016).
- *Adversarial perturbations to the final point*, where the final recommendation x may be perturbed up to some level δ (Bertsimas et al., 2010; Bogunovic et al., 2018).
- *Adversarial perturbations to samples*, where the observations are adversarially corrupted up to some maximum budget (Bogunovic et al., 2020a).

These methods are primarily focused on proposing methods that defend against the proposed uncertainty model to improve robustness for GP optimization. There has been minimal work studying the problem from an attacker's perspective in the literature, i.e., what kinds of attacks would be successful against non-robust algorithms. We examine such a perspective in this work, focusing on adversarial perturbations to samples (Bogunovic et al., 2020a).

[1]School of Computing, National University of Singapore [2]Department of Mathematics & Institute of Data Science, National University of Singapore. Correspondence to: Eric Han <eric_han@nus.edu.sg>, Jonathan Scarlett <scarlett@comp.nus.edu.sg>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

Our study is related to that of attacks on stochastic linear bandits (Garcelon et al., 2020), but we move to the GP setting which is inherently non-linear and poses substantial additional challenges. We focus in particular popular algorithms into selecting from a specific set of (typically suboptimal) *target actions* as much as possible, using GP-UCB (Srinivas et al., 2010) and a related elimination-based algorithm as representative examples.

Prior to (Garcelon et al., 2020), analogous works studied attacks and defenses for multi-armed bandits (Jun et al., 2018; Lykouris et al., 2018). These are less related here, since they assume finite domains with independent arms.

1.2. Contributions

The main contributions of this paper are as follows:

1. We theoretically characterize conditions under which an adversarial attack can succeed against GP-UCB or elimination even with an attack budget far smaller than the time horizon, both in the cases of the function being known (white-box attack) and unknown (black-box attack) to the attacker.
2. We present various attacks inspired by our analysis:
   (a) with knowledge of the function: Subtraction Attack (two variants), Clipping Attack.
   (b) without knowledge of the function: Aggressive Subtraction Attack (two variants).

   We demonstrate the effectiveness of these attacks via experiments on a diverse range of objective functions.

More broadly, we believe that our work fills an important gap in the literature by moving beyond finite domains and/or linear functions, and giving the first study of robust GP bandits from the attacker's perspective.

2. Setup

We consider the setup proposed in (Bogunovic et al., 2020a), described as follows. The player seeks to maximize an unknown function $f(\mathbf{x})$ over $\mathbf{x} \in D$, and we model the smoothness of f by assuming that it has RKHS norm at most B according to some kernel k, i.e., $f \in \mathcal{F}_k(B)$ where $\mathcal{F}_k(B) = \{f : \|f\|_k \leqslant B\}$. We focus primarily on the widely-adopted Matérn kernel, defined as

$$k_{\text{Mat}}(x, x') = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}\|x - x'\|}{l} \right)^{\nu} J_\nu \left(\frac{\sqrt{2\nu}\|x - x'\|}{l} \right), \tag{1}$$

where $l > 0$ denotes the length-scale, $\nu > 0$ is the smoothness parameter, and J_ν denotes the modified Bessel function. This kernel provides useful properties that we can exploit in our theoretical analysis, and is also one of the most widely-adopted kernels in practice. In addition, it closely resembles the squared exponential (SE) kernel in the case that ν is large (Rasmussen, 2006).

In the non-corrupted setting (e.g., see (Srinivas et al., 2010; Chowdhury & Gopalan, 2017)), the player samples $\mathbf{x}_t$ and observes $y_t = f(\mathbf{x}_t) + z_t$, where $z_t \sim \mathcal{N}(0, \sigma^2)$ is random noise. In the presence of adversarial noise, we consider corrupted observations taking the form

$$y_t = f(\mathbf{x}_t) + c_t + z_t, \tag{2}$$

where $z_t \sim \mathcal{N}(0, \sigma^2)$, and c_t is adversarial noise injected at time t. We consider c_t as being chosen by an *adversary* or *attacker*. In order to make the problem meaningful, the adversary's power should be limited, so we constrain

$$\sum_{t=1}^{n} |c_t| \leqslant C \tag{3}$$

for some total corruption budget C. Following (Bogunovic et al., 2020a), we adopt the same definition of regret as the uncorrupted setting: $R_T = \sum_{t=1}^{T} r_t$, where $r_t = f(\mathbf{x}^*) - f(\mathbf{x}_t)$, and where $\mathbf{x}^*$ is any maximizer of f.

We note that c_t could also potentially be considered as part of the objective when defining regret; the two notions are closely related, and differ by at most $O(C)$ (Lykouris et al., 2018; Bogunovic et al., 2020a). For all of our results, this connection ensures that a successful attack (i.e. linear regret in T) with respect to the above notion implies the same with respect to the alternative regret notion.

2.1. Knowledge Available to the Adversary

Naturally, the ability to attack/defend in the preceding setup may vary significantly depending on what is assumed to be known to the adversary. For instance, the adversary may or may not know f, know $\mathbf{x}_t$, know which algorithm the player is using, and so on. In addition, as noted in the literature on robust bandit problems (e.g., (Lykouris et al., 2018; Bogunovic et al., 2020b)), one may consider the case that the player can randomize $\mathbf{x}_t$, and the adversary knows the distribution but not the specific choice.[1]

In this paper, our focus is on attacking widely-adopted deterministic algorithms such as GP-UCB (Srinivas et al., 2010), so we do not consider such randomization. In this case, knowing the "distribution of $\mathbf{x}_t$" becomes equivalent to knowing $\mathbf{x}_t$, and we assume that such knowledge is available throughout the paper. We consider both the cases of f being known and unknown to the attacker.

2.2. Targeted vs. Untargeted Attacks

At this stage, we find it useful to distinguish between two types of attack. In an *untargeted attack*, the adversary's

[1] In such a case, c_t in (3) is typically replaced by $\max_{\mathbf{x}} |c_t(\mathbf{x})|$, where $c_t(\cdot)$ is the adversary's corruption function at time t.

sole aim is to make the player's cumulative regret as high as possible (e.g., $R_T = \Omega(T)$). In contrast, in a *targeted attack*, the adversary's goal is to make the player choose actions in a particular region $\mathcal{R}_{\text{target}} \subseteq D$. This generalizes the finite-arm notion of seeking to make the player pull a particular target arm (Jun et al., 2018).

Of course, a targeted attack can also be used to ensure a large regret: If $\mathcal{R}_{\text{target}}$ satisfies the property that every $\mathbf{x} \in \mathcal{R}_{\text{target}}$ has $r(\mathbf{x}) = \Omega(1)$, then any attack that forces $\Omega(T)$ selections in $\mathcal{R}_{\text{target}}$ also ensures $R_T = \Omega(T)$.

More generally, to assess the performance of a targeted attack, we define the quantity after T rounds

$$N_T^{\text{target}} = \sum_{t=1}^{T} \{\mathbf{x}_t \in \mathcal{R}_{\text{target}}\}, \tag{4}$$

counting the number of arm pulls within the target region.

3. Theoretical Study

This section introduces some attacks and gives conditions under which they provably succeed even when the budget C is small compared to the time horizon T. These results are not only of interest in their own right, but will also motivate several of our other attacks (without theory) in Sec. 4.

Motivated by successful attacks on bandit problems (Jun et al., 2018; Garcelon et al., 2020), we adopt the idea of perturbing the function *outside* the $\mathcal{R}_{\text{target}}$ in a manner such that the perturbed function's maximizer is in the $\mathcal{R}_{\text{target}}$. Then, a (non-robust) optimization algorithm will steer towards $\mathcal{R}_{\text{target}}$ and stay there, with any points sampled in $\mathcal{R}_{\text{target}}$ remaining unperturbed.

This idea is somewhat trickier to implement in the GP bandit setting than the finite-arm or linear bandit setting, and the details are given below.

3.1. Optimization Algorithms

Our attack methods can be applied to any GP bandit algorithm, and Thm. 1 below states general conditions under which the attack succeeds. As two specific examples, we will consider the following widely-used algorithms that are representative of broader techniques in the literature:

- *GP-UCB* (Srinivas et al., 2010): The t-th point is selected according to

$$\mathbf{x}_t = \arg\max_{\mathbf{x} \in D} \mu_{t-1}(\mathbf{x}) + \beta_t^{1/2} \sigma_{t-1}(\mathbf{x}), \tag{5}$$

  for some suitably-chosen exploration parameter β_t, where $\mu_{t-1}(\cdot)$ and $\sigma_{t-1}(\cdot)$ are the posterior mean and standard deviation after sampling $t-1$ points (Rasmussen, 2006).

- *MaxVar + Elimination* (Contal et al., 2013): At each time, define the set of *potential maximizers* M_t to contain all points whose UCB (i.e., $\mu_{t-1}(\mathbf{x}) + \beta_t^{1/2}\sigma_{t-1}(\mathbf{x})$) is at least as high as the highest LCB (i.e., $\mu_{t-1}(\mathbf{x}) - \beta_t^{1/2}\sigma_{t-1}(\mathbf{x})$). Then, select the point in M_t with the highest posterior variance.

For both of these algorithms, the exploration parameter β_t is set to the value used in theoretical studies (e.g., (Chowdhury & Gopalan, 2017)):

$$\beta_t^{1/2} = B + \sigma\lambda^{-1/2}\sqrt{2(\gamma_{t-1} + \ln(1/\delta))}. \tag{6}$$

where λ is a free parameter (e.g., $\lambda = 1$), δ is the target error probability, and γ_t is the maximum information gain at time t (Srinivas et al., 2010). The latter quantity scales as $\gamma_t = \tilde{O}\big(T^{\frac{d}{2\nu+d}}\big)$ for the Matérn kernel (Vakili et al., 2020).

The reason for focusing on these algorithms is that they have well-known guarantees on the regret in the uncorrupted setting, and such guarantees turn out to be a key ingredient in establishing the success of attacks in the corrupted setting. While these algorithms are far from exhaustive, they serve as representative examples of general non-robust algorithms.

3.2. Conditions for a Successful Attack

Our main theoretical result motivating our attacks is stated as follows. This result applies to *any* algorithm that has non-robust guarantees of not playing too many strictly suboptimal actions. The asymptotic notation $o(1)$ is defined with respect to the limit $T \to \infty$.

Theorem 1. *Consider an adversary that performs an attack shifting the original function $f \in \mathcal{F}_k(B)$ to another function $\tilde{f}$ (i.e., set $c_t = \tilde{f}(\mathbf{x}_t) - f(\mathbf{x}_t)$ at time t as long as the corruption budget permits it). Suppose that the following properties hold for some $\Delta > 0$ and $B_0 > 0$:*

(i) For any $\mathbf{x}$ that is Δ-optimal for $\tilde{f}$, it holds that both $\mathbf{x} \in \mathcal{R}_{\text{target}}$ and $\tilde{f}(\mathbf{x}) = f(\mathbf{x})$;
(ii) For all $\mathbf{x} \in D$, it holds that $|f(\mathbf{x}) - \tilde{f}(\mathbf{x})| \leqslant B_0$;
(iii) It holds that $\|\tilde{f}\|_k \leqslant B$ (i.e., $\tilde{f} \in \mathcal{F}_k(B)$).

In addition, suppose that in the absence of an adversary, the optimization algorithm guarantees, for any $f \in \mathcal{F}_k$ and with probability at least $1-\delta$, that at most N_0 played actions are Δ-suboptimal, for some N_0 depending on (T, Δ, δ).

Then, in the presence of the adversary, with probability at least $1-\delta$, the attack succeeds in forcing $T(1-o(1))$ actions to be played from $\mathcal{R}_{\text{target}}$, while using an attack budget C of at most $B_0 N_0$.

Proof. By the first assumed property, whenever the algorithm plays actions that are Δ-optimal *with respect to* $\tilde{f}$, it holds that $c_t = 0$, and no budget is spent. The third

property allows us to bound the number of actions failing to satisfy such a property by N_0, where this bound holds with probability at least $1-\delta$ by the assumption on the algorithm. The second property ensures that each action uses an amount of budget satisfying $|c_t| \leqslant B_0$, and the result follows. □

Combining this result with recently-established results from (Cai et al., 2021), we obtain the following corollary for the GP-UCB and elimination algorithms.

Corollary 1. *Under the setup of Thm. 1, with probability at least $1-\delta$ (for δ used in the algorithms), the attack succeeds in forcing $T(1-o(1))$ actions to be played from $\mathcal{R}_{\text{target}}$ against GP-UCB (respectively, the elimination algorithm) using an attack budget C of at most $B_0 N_{\max}(\Delta, T)$ (respectively, $B_0 N'_{\max}(\Delta)$), where*

$$N_{\max}(\Delta, T) = \max\Big\{N : N \leqslant \frac{C_1 \gamma_N \beta_T}{\Delta^2}\Big\}, \quad (7)$$

$$N'_{\max}(\Delta) = \max\Big\{N : N \leqslant \frac{4C_1 \gamma_N \beta_N}{\Delta^2}\Big\}, \quad (8)$$

with $C_1 = \frac{8\lambda^{-1}}{\log(1+\lambda^{-1})}$.

Proof. This result follows readily by combining Thm. 1 with existing bounds on the number of Δ-suboptimal actions (i.e., having $f(\mathbf{x}) < \max_{\mathbf{x}'} f(\mathbf{x}') - \Delta$) from (Cai et al., 2021); in the standard (non-adversarial) GP bandit setting, we have the following:

(i) For GP-UCB, the number of actions chosen in T rounds that are Δ-suboptimal is at most $N_{\max}$ with probability at least $1-\delta$.
(ii) For the elimination algorithm, this can further be reduced to $N'_{\max}$.

□

While the preceding results are phrased as though the adversary had perfect knowledge of f, we will highlight further special cases below where this need not be the case.

As discussed in (Cai & Scarlett, 2020), as long as the smoothness parameter ν is not too small, $N_{\max}(\Delta, T))$ scales slowly with T (e.g., $\sqrt{T}$ or $T^{0.1}$), and $N'_{\max}(\Delta)$ has no dependence on T at all. Hence, for such sufficiently smooth functions with the Matérn kernel, Thm. 1 gives conditions under which an attack succeeds with a far smaller budget than the time horizon. Note that as ν grows large for the Matérn kernel, $N_{\max}$ behaves as $\frac{T^{\epsilon'}}{\Delta^{2+\epsilon}}$ for some small ϵ and ϵ', and $N'_{\max}$ behaves as $\frac{1}{\Delta^{2+\epsilon}}$ for some small ϵ.

Moreover, in view of Thm. 1, the stronger the guarantee on the number of Δ-suboptimal actions in the uncorrupted setting, the smaller the attack budget will be in a counterpart of Cor. 1. For instance, if we were to attack the recent *batched* elimination algorithm of (Li & Scarlett, 2021), the required budget would be sublinear in T for all $\nu > \frac{1}{2}$, regardless of d. However, we prefer to focus on the above non-batched algorithm, since it simpler and more standard in the literature.

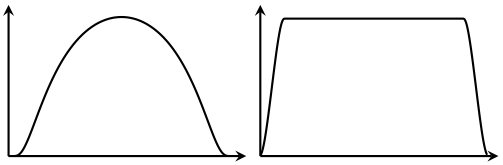


Figure 1: Illustration of the various function constructions – bump function (left) and convolved bump function (right).

Sufficiency vs. Necessity. We note that Thm. 1 only states sufficient conditions for a successful attack; analogous necessary conditions are difficult, and appear to be absent even in simpler settings studied previously (e.g., linear bandits). We note that condition (ii) is not always necessary, because the two functions could differ drastically at some far-away suboptimal point that is never queried. Condition (iii) is also not always necessary, because Clipping succeeds without it. We believe that condition (i) is "closest to necessary", but even so, although the attacker's budget would get exhausted without it, it could be the case that the "damage has already been done", and the attack still succeeds.

The Role of RKHS Norm. As noted by a reviewer, the role of RKHS on the attack success is subtle. On the one hand, a higher RKHS norm could be associated with more local optima that the attacker can exploit. Furthermore, it could additionally help the attacker if they can perturb f using a constant fraction of the total budget B. On the other hand, if one asks which functions are the hardest to attack, then a higher B implies more flexibility in coming with such a function. In general, we expect that there is no direct correspondence between B and the attack difficulty; in particular, among functions with a large RKHS norm, some are easier to attack, and some are harder.

3.3. Applications to Specific Scenarios

Before we discuss two natural approaches to ensuring the conditions in Thm. 1, we describe some useful function constructions from the existing literature (see Fig. 1):

- *Bump function in spatial domain*: It is known from (Bull, 2011; Cai & Scarlett, 2020) that the function

$$h(\mathbf{x}) = \exp\left(\frac{-1}{1-\|\mathbf{x}\|^2}\right)\mathbf{1}\{\|\mathbf{x}\| \leqslant 1\}, \quad (9)$$

has bounded RKHS norm under the Matérn kernel, and is non-negative with bounded support and maximum

at $\mathbf{x} = \mathbf{0}$. Moreover, we can form a scaled version of h with height ϵ and support of radius w, and the resulting RKHS norm scales as $O\big(\frac{\epsilon}{w^\nu}\big)$ (i.e., it can be made at most B with $w = \Theta\big(\big(\frac{\epsilon}{B}\big)^{1/\nu}\big)$).

- *Convolved bump function*: Following (Cai & Scarlett, 2020), we consider taking the width-w height-1 bump function from the previous dot point, and convolving it with a function that equals 1 for $\mathbf{x} \in S$, and 0 for $\mathbf{x} \notin S$, where S is some compact subset of $\mathbb{R}^d$. Note that in (Cai & Scarlett, 2020, Lemma 6), S was ball-shaped, but the analysis extends to the general case. Then, after suitable scaling to make the new function's maximum height equal to some value τ, we obtain a function $g(\mathbf{x})$ satisfying the following:
  - If the ball of radius w around $\mathbf{x}$ is contained in S, then $g(\mathbf{x}) = \tau$;
  - If the ball of radius w around $\mathbf{x}$ is completely outside S, then $g(\mathbf{x}) = 0$;
  - In all cases in between these, $g(\mathbf{x}) \in (0, \tau)$.

  Moreover, the RKHS norm satisfies $\|g\|_k \leqslant O\big(\frac{\tau \cdot \mathrm{vol}(S)}{w^\nu}\big)$ (Cai & Scarlett, 2020), so if $\mathrm{vol}(S) = O(1)$ we can ensure an RKHS norm of at most B with a choice of the form $w = \Theta\big(\big(\frac{\tau}{B}\big)^{1/\nu}\big)$.

We now discuss two general attack approaches, which we will build on later in Sec. 4.

Approach 1. To guarantee that $\|\tilde{f}\|_k \leqslant B$, it is useful to adopt the decomposition

$$\|\tilde{f}\|_k \leqslant \|f\|_k + \|\tilde{f} - f\|_k, \tag{10}$$

which follows from the triangle inequality. Hence, if the "RKHS norm budget" B is not entirely used up by f (e.g., $\|f\|_k = \frac{B}{2}$), then we can maintain $\|\tilde{f}\|_k \leqslant B$ by setting

$$\tilde{f}(\mathbf{x}) = f(\mathbf{x}) - h(\mathbf{x}) \tag{11}$$

for some $h(\mathbf{x})$ with suitably bounded norm (e.g., $\|h\|_k \leqslant \frac{B}{2}$). The convolved bump function described above (which approximates a rectangular function) is useful for constructing h.

This approach is immediately applicable to the case that the interior of $\mathcal{R}_{\mathrm{target}}$ contains a local maximum, e.g., see Fig. 3. In this case, we can simply form bumps to "swallow" any higher maxima (or maxima within Δ of the target peak). As long as the convolved bump functions used for this purpose have a small enough height and transition width to maintain that $\|h\|_k \leqslant B - \|f\|_k$, the resulting function will satisfy the conditions of Thm. 1.

However, finding the precise locations of such bumps and setting their parameters may be difficult, or even entirely impossible if f is unknown, even if there is plenty of RKHS norm budget that can be utilized.

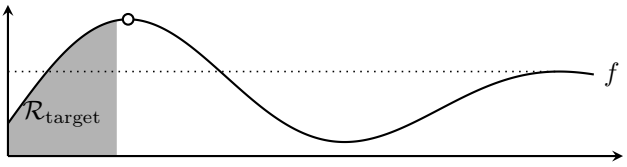


Figure 2: Difficult case where f is increasing in $\mathcal{R}_{\mathrm{target}}$.

Approach 2. In view of the above discussion, we propose a more aggressive approach that seeks to push the function values downward equally (or approximately equally) for *all* points outside $\mathcal{R}_{\mathrm{target}}$. This can be done by taking a convolved bump function that covers the entire domain with some height $h_{\max}$ (i.e., $h(\mathbf{x}) = -h_{\max}$), but then "re-adding" a bump that covers $\mathcal{R}_{\mathrm{target}}$. Depending on which is more convenient, the transition region could lie either inside or outside $\mathcal{R}_{\mathrm{target}}$. See Fig. 5 for an illustration.

Once again, the preceding attack can be applied while maintaining the conditions Thm. 1 in cases where $\mathcal{R}_{\mathrm{target}}$ contains a local maximum, provided that the bumps used in its construction do not exceed the RKHS norm budget.

Example Difficult Case. To highlight the consideration of cases with local maxima in $\mathcal{R}_{\mathrm{target}}$ above, consider the 1D (counter-)example in Fig. 2. Here, there is no apparent way to construct $h(\mathbf{x})$ to satisfy the conditions of Thm. 1; the function continues to increase when leaving $\mathcal{R}_{\mathrm{target}}$, and performing any *smooth* perturbation to the function to push those values downward would either keep the maximizer outside $\mathcal{R}_{\mathrm{target}}$, or amount to having Δ-optimal points for $\tilde{f}$ such that $\tilde{f}(\mathbf{x}) \neq f(\mathbf{x})$. Despite this difficulty, we will see that our attacks can still be effective in such situations experimentally, albeit requiring a larger attack budget.

4. Attack Methods

4.1. Overview

In this section, we introduce our main proposed adversarial attacks on GP bandits. Based on Thm. 1, we adopt the idea of perturbing the function f to construct another function $\tilde{f}$ such that ideally the following properties hold for some set $\mathcal{R}_0 \subseteq \mathcal{R}_{\mathrm{target}}$ (often taking the full set, $\mathcal{R}_0 = \mathcal{R}_{\mathrm{target}}$):

(i) $\tilde{f}(\mathbf{x}) = f(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{R}_0$;
(ii) All maximizers of $\tilde{f}$ lie inside $\mathcal{R}_0$, and ideally all points outside $\mathcal{R}_{\mathrm{target}}$ are suboptimal by at some strictly positive value $\Delta > 0$;
(iii) $\tilde{f}$ satisfies the RKHS norm constraint, i.e., $\|\tilde{f}\|_k \leqslant B$.

We use the Synthetic1D objective function in (16) to illustrate the attack methods in Fig. 3-5.

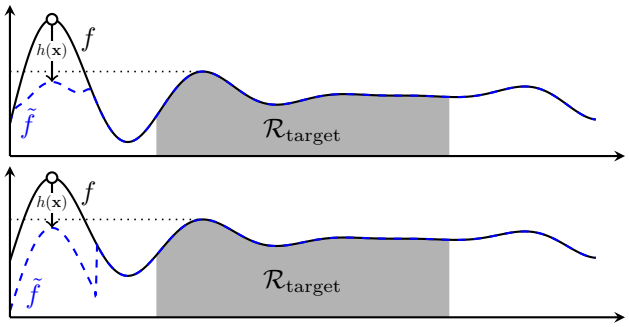


Figure 3: Subtraction Attack for known f with different $h(\mathbf{x})$ – Subtraction Rnd (top) and Sq (bottom).

4.2. Subtraction Attack (Known f)

For this attack, we follow the ideas discussed in Approach 1 of Sec. 3.3. To maintain property (iii), i.e., $\tilde{f} \in \mathcal{F}_k(B)$, we assume that $f \in \mathcal{F}_k(B/2)$,[2] and seek to find a perturbation function $h \in \mathcal{F}_k(B/2)$ such that

$$\tilde{f}(\mathbf{x}) = f(\mathbf{x}) - h(\mathbf{x}) \tag{12}$$

satisfies properties (i) and (ii). The triangle inequality then gives that $\|\tilde{f}\|_k \leqslant B$, as desired for property (iii).

We let $h(\mathbf{x})$ be a sum of support-bounded, with the support lying entirely outside $\mathcal{R}_0$ designed to "swallow" the peaks outside $\mathcal{R}_0$. For Subtraction Rnd we construct these functions using the bump function $h_{\text{bump}}(\mathbf{x}) = \exp\left(\frac{-1}{1-\|\mathbf{x}\|^2}\right)\mathbf{1}\{\|\mathbf{x}\| \leqslant 1\}$ (see Sec. 3.3 for details), whereas for Subtraction Sq we use the simpler indicator function $h_{\text{ind}}(\mathbf{x}) = \mathbf{1}\{\|\mathbf{x}\| \leqslant 1\}$. Examples are shown in Fig. 3.

Based on Sec. 3, Subtraction Rnd has strong theoretical guarantees under fairly mild assumptions, but a disadvantage is requiring knowledge of f. Moreover, there may be many ways to construct h satisfying the requirements given, and finding a good choice may be difficult, particularly in higher dimensions where the function cannot be visualized.

4.3. Clipping Attack (Known f)

The subtraction attack in Sec. 4.2 was developed for the primary purpose of obtaining theoretical guarantees, but in practice, finding $h(\cdot)$ with the desired properties may be challenging. Here we provide a more practical attack without theoretical guarantees, but with a similar general idea. Specifically, we propose to directly attain properties (i) and (ii) above by setting

$$\tilde{f}(\mathbf{x}) = \begin{cases} f(\mathbf{x}) & \mathbf{x} \in \mathcal{R}_{\text{target}} \\ \min\{f(\mathbf{x}), f(\tilde{\mathbf{x}}^*) - \Delta\} & \mathbf{x} \notin \mathcal{R}_{\text{target}}, \end{cases} \tag{13}$$

[2]The factor $\frac{1}{2}$ can be replaced by other constants in $(0, 1)$.

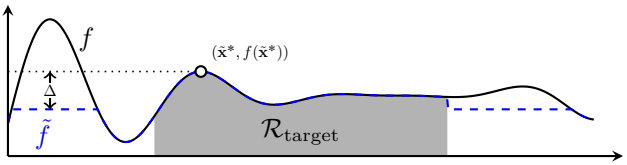


Figure 4: Clipping attack for known f.

where $\tilde{\mathbf{x}}^* = \arg\max_{\mathbf{x} \in \mathcal{R}_{\text{target}}} f(\mathbf{x})$ (illustrated in Fig. 4).

Here, unlike for the subtraction attack, property (iii) does not hold, which is why the theoretical analysis does not follow. Despite this disadvantage (and the requirement of knowing f), this attack has the clear advantage of being simple and easy to implement, and we will see in Sec. 5 that it can be highly effective in experiments.

4.4. Aggressive Subtraction Attack (Unknown f)

In the case that f is unknown, we propose to build on the idea of the subtraction attack, but to be highly aggressive and subtract *all* points outside $\mathcal{R}_{\text{target}}$ by roughly the same value $h_{\max}$. This is a special case of (12), but here we consider $h(\cdot)$ having a much wider support than described above, so we find it best to highlight separately.

To ensure that $\|h\|_k \leqslant \frac{B}{2}$, we need to consider a "transition region" where $h(\cdot)$ transitions from zero to $h_{\max}$. Overall, we are left with $h(\mathbf{x})$ equaling zero within $\mathcal{R}_0$, $h_{\max}$ outside $\mathcal{R}_{\text{target}}$, and intermediate values within $\mathcal{R}_{\text{target}} \backslash \mathcal{R}_0$.

Again based on Sec. 3, this attack has strong theoretical guarantees. A notable advantage is not requiring precise knowledge of f; it suffices that $\mathcal{R}_{\text{target}}$ has a suitable local maximum and $h_{\max}$ is large enough, but no specific details of the function need to be known. In addition, this attack is fairly straightforward to implement, only requiring the selection of $h_{\max}$ and a transition region width.

A disadvantage is that a higher budget C may be needed in practice due to being highly aggressive. On the other hand, similar aggressive approaches have shown success in related bandit problems (Jun et al., 2018; Garcelon et al., 2020).

Simplified Aggressive Subtraction Attack. In the same way that Subtraction Sq simplifies Subtraction Rnd, we can simplify the aggressive subtraction attack as follows, for some $h_{\max} > 0$:

$$\tilde{f}(\mathbf{x}) = \begin{cases} f(\mathbf{x}) & \mathbf{x} \in \mathcal{R}_{\text{target}} \\ f(\mathbf{x}) - h_{\max} & \mathbf{x} \notin \mathcal{R}_{\text{target}}. \end{cases} \tag{14}$$

This can again be implemented without knowledge of f.

Similar to above, we would like to choose $h_{\max}$ large enough so that the maximizer of $\tilde{f}$ is in $\mathcal{R}_{\text{target}}$. This again has the disadvantage of potentially requiring a large

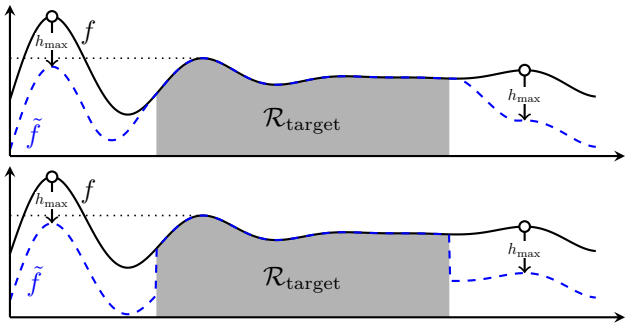


Figure 5: Aggressive Subtraction attack for unknown f – with "transition region" (top) and without (bottom).

corruption budget, but compared to the clipping attack, we gain the advantage of not needing to know f.

Both versions of Aggressive Subtraction are illustrated in Fig. 5. In the rest of the paper, whenever we mention this attack, we are referring to the simplified variant.

5. Experimental Results

For convenience, our experiments[3] consider targeted attacks and do not constrain the attack to any particular budget; we instead explore the trade-off between attack success rate (i.e., fraction of actions played that fall in the $\mathcal{R}_{\text{target}}$) and attack cost (e.g., sum of perturbation levels used).

To the best of our knowledge, our work is the first for this setting, so there are no existing attacks to compare against. Hence, our focus is only comparing our proposed attacks to each other (i.e. Subtraction Rnd, Subtraction Sq, Clipping, Aggressive Subtraction) along with two trivial baselines. The baselines are Random, which perturbs the function evaluation with $\mathcal{N}(\mu_a, {\sigma_a}^2)$ and, No Attack which does not perturb the function evaluation at all.

We exclude Subtraction Rnd, Subtraction Sq from experiments on higher dimensional functions (i.e. $\geqslant 3$), due to the difficulty discussed in Sec. 4.2.

We compare the attacks on a variety of objective functions of up to 6 dimensions. For the experiments on synthetic functions, we manually add $z_t \sim \mathcal{N}(0, 0.01^2)$ noise. Detailed experimental details are given in Sec. A (appendix) and our complete code is included in the supplementary.

5.1. Setup

In order to understand the behavior of each attack, we run each attack 300 times, varying 30 different hyperparameter choices (i.e. $\Delta, h_{\max}, (\mu_a, \sigma_a), h(\mathbf{x})$) with 10 differing conditions (initial points, instances of the objective function, and random seeds). For both the Subtraction Rnd and Subtraction Sq attacks, we vary only the y-transformation hyperparameter, which we refer to as $h_{\max}$. The different hyperparameter settings are chosen such that, as much as possible, we aim to cover the entire spectrum of the behavior (starting with a lower success rate and ending with a high success rate) of the attack method for each experiment. We run 100 or 250 optimization steps depending on the dimensionality, and adopt the Matérn $5/2$ kernel to match our theory; see Sec. A.3 and Sec. A.5 for further details.

We focus on attacking the GP-UCB algorithm and defer the elimination algorithm to Sec. A (appendix), since GP-UCB is much more widespread and Cor. 1 indicates that it should be the harder algorithm to attack. (One reason for this is that MaxVar explicitly removes points from further consideration, so it can be forced to permanently discard the true optimum.) Following typical choices adopted in existing works' experiments (e.g., (Srinivas et al., 2010; Rolland et al., 2018)), we select the exploration parameter as $\beta_t = 0.5 \log(2t)$. There are very few defense algorithms in the literature that are appropriate here; in particular, the main algorithm in (Bogunovic et al., 2020a) (Fast-Slow GP-UCB) was introduced for theoretical purposes and seemingly not intended to be practical. However, (Bogunovic et al., 2020a) also shows that GP-UCB with *larger choices of* β_t *depending on* C can be provably more robust, and we provide experimental support for this in Sec. A.8.

5.2. Metrics

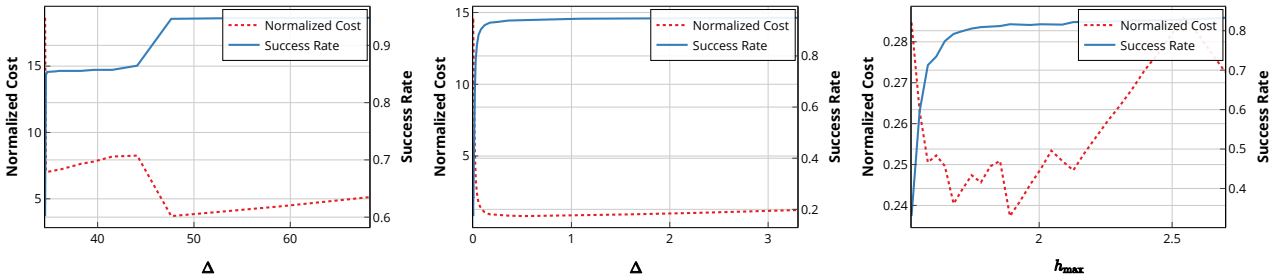
We measure the following up to iteration t, with $X_t = (\mathbf{x}_1, \ldots, \mathbf{x}_t)$ and adversarial noise $A_t = (|c_1|, \ldots, |c_t|)$:

- Success-Rate $(t) = \frac{|\mathcal{R}_{\text{target}} \cap X_t|}{t}$ is the proportion of actions played that lie within $\mathcal{R}_{\text{target}}$.
- Normalized-Cost $(t) = \sum_{a \in A_t} \frac{a}{f_{\max} - f_{\min}}$ is the sum over the history of adversarial noise, normalized by the function range for comparison across experiments.

5.3. Synthetic Experiments

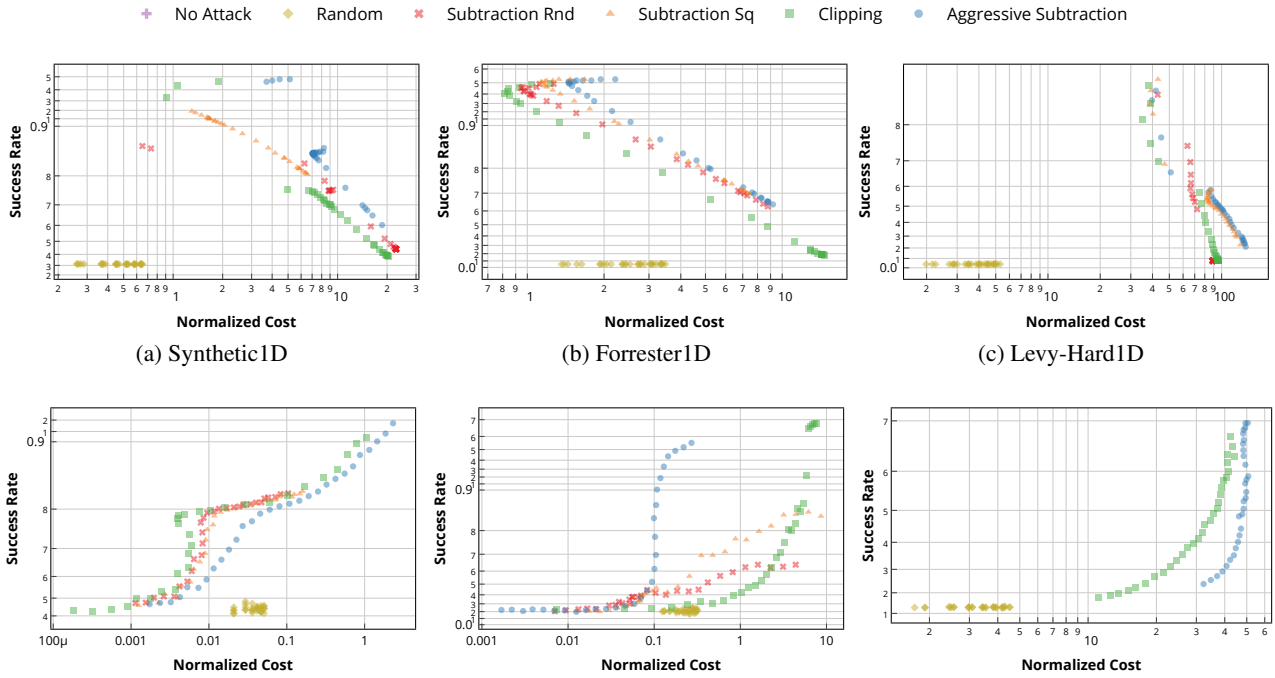
The synthetic functions are described and plotted in Sec. A.2 (appendix). The performance of each attack method depends on the associated hyperparameter (e.g., $h_{\max}$ or Δ). Ideally, we want the best hyperparameter for the attack where it spends the least cost for the highest success rate. One might expect that a higher cost is always associated with a higher success rate, but this is not quite the case, as we observe in Fig. 6. The subtlety is that a poorly-chosen hyperparameter can be poor in both regards. Typically, what happens is that when the hyperparameter is at its 'boundary value' (e.g., $\Delta = 0$ in Fig. 4) the attack is not very successful, because GP-UCB explores both the clipped regions and $(\tilde{\mathbf{x}}^*, f(\tilde{\mathbf{x}}^*))$.

[3]The code is available at `https://github.com/eric-vader/Attack-BO`.



(a) Aggressive Subtraction on Synthetic1D (b) Clipping on Forrester1D (c) Subtraction Rnd on Levy1D

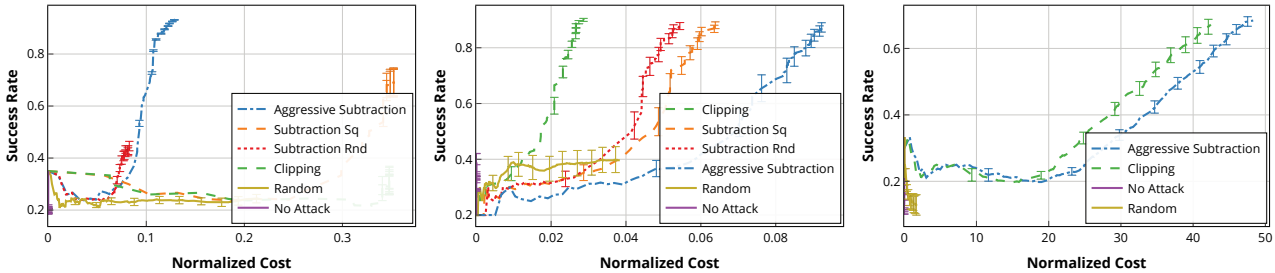
Figure 6: Effects of hyperparameters on the different attacks.



(a) Synthetic1D (b) Forrester1D (c) Levy-Hard1D

(d) Camelback2D (e) Bohachevsky2D (f) Robot3D

Figure 7: Scatter plots of runs averaged over random seeds for various functions.



(a) Bohachevsky2D (b) Branin2D (c) Robot3D

Figure 8: Plots of each attack with the most efficient (best success rate over normalized cost) hyperparameter.

However, going carefully beyond the boundary value can quickly lead to a near-maximal success rate with a fairly modest cost. Perhaps less intuitively, even attacking more aggressively (e.g., increasing Δ) sometimes maintains a similar cost-success trade-off, suggesting that it may be preferable to be 'over-aggressive' than 'under-aggressive'.

From Fig. 7, we observe that Clipping performs consistently well across the various functions, providing competitive success rates and costs. Both Subtraction Rnd and Subtraction Sq tend to have a performance 'in between' that of Clipping and Aggressive Subtraction. Due to Subtraction Rnd's smooth $h(\mathbf{x})$, Subtraction Rnd tends to narrowly beat Subtraction Sq in most experiments, e.g., see Fig. 7b. The trend is also observed in Levy1D, the easier version of Levy-Hard1D, as seen in Fig. 16c (appendix). However, these aforementioned methods require prior knowledge of f, whereas Aggressive Subtraction works well in practice without knowledge of f.

We additionally note in Fig. 7 that across all of the functions considered, there typically exists a successful attack with a relatively low normalized cost, e.g., on the order of magnitude 1 or less, though more difficult cases also exist (e.g., Levy-Hard1D). This highlights the general lack of robustness of standard solutions to the GP bandit problem.

In Fig. 8, we plot the performance of each attack method with its most efficient hyperparameter. The hyperparameter with the largest success rate over cost is the most efficient,

$$\theta_{\text{efficient}} = \arg\max_{\theta} \frac{\text{Success-Rate}\,(T)_\theta}{\text{Normalized-Cost}\,(T)_\theta}. \tag{15}$$

Considering the most efficient hyperparameter, Clipping continues to be successful and cost effective compared to others for most functions. Subtraction Rnd and Subtraction Sq tend to perform in between Clipping and Aggressive Subtraction, e.g., see Fig. 8b.

There are also some interesting cases where this trend is not observed; Subtraction Rnd and Subtraction Sq struggle to achieve a high success rate in Camelback2D, while Aggressive Subtraction and Clipping achieve a higher success rate after spending more cost. Camelback2D has a symmetric surface with multiple global and local optima in different regions, which increases the difficulty in constructing $h(\mathbf{x})$ for the subtraction attacks. Similarly, for the attacks with the most efficient hyperparameter in Bohachevsky2D, Aggressive Subtraction is the most efficient, spending the least but having the highest success rate. Similar findings are observed in Fig. 7e. Bohachevsky2D has a bowl shape with a single maximum, which is easy to optimize but hard to attack.

As a result, Bohachevsky2D increases throughout $\mathcal{R}_{\text{target}}$ and when leaving $\mathcal{R}_{\text{target}}$, causing the attack to be especially difficult as explained in Sec. 3.3 with Fig. 2. Hence, there is no obvious way to construct $h(\mathbf{x})$ for both Subtraction Rnd and Subtraction Sq (also for Camelback2D). We believe that Clipping is less effective because it creates $\tilde{f}$ which is largely flat (equal to the clipped value), the algorithm may spend too long exploring this flat region. In contrast, Aggressive Subtraction maintains the bowl shape (shifted downwards) outside $\mathcal{R}_{\text{target}}$, which the algorithm can more readily stop exploring due to believing it to be suboptimal. Hence, Aggressive Subtraction performs best here in terms of success rate and cost.

5.4. Robot Pushing Experiments

We consider the deterministic robot pushing objective from (Wang & Jegelka, 2017), to find the best pre-image to push an object towards a target location. We test both the 4-dimensional variant $f_4(r_x, r_y, r_t, r_\theta)$ and 3-dimensional variant f_3, where $r_\theta = \arctan\left(r_y / r_x\right)$. Both functions return the distance from the pushed object to the target location. Here, $r_x, r_y \in [-5, 5]$ is the robot location, pushing duration $r_t \in [1, 30]$ and pushing angle $r_\theta \in [0, 2\pi]$. From Fig. 7f and Fig. 8c, the results here exhibit similar findings to most results on synthetic experiments; in particular, Clipping performs better than Aggressive Subtraction.

5.5. Further Experiments

In Sec. A (appendix), we present more details on the above findings, as well as exploring (i) different kernel choices, (ii) attacking the elimination algorithm, (iii) online vs. offline kernel learning, (iv) a more robust variant of GP-UCB from (Bogunovic et al., 2020a), and (v) a proof-of-concept attack that automatically adapts its hyperparameters.

6. Conclusion

We have studied the problem of adversarial attacks on GP bandits, providing both a theoretical understanding of when certain attacks are guaranteed to succeed, as well as a detailed experimental study of the cost and success rate of our attacks. Possible future research directions include (i) further investigating more "automated" attacks that can adapt their hyperparameters online, and (ii) theoretical lower bounds on the attack budget, which are currently unknown even in the simpler linear bandit setting.

Acknowledgments.

This work was supported by the Singapore National Research Foundation (NRF) under grant number R-252-000-A74-281.

References

Anaconda. Anaconda software distribution, 11 2016. URL https://docs.anaconda.com/.

Beland, J. J. and Nair, P. B. Bayesian optimization under uncertainty. NIPS BayesOpt 2017 workshop, 2017.

Bertsimas, D., Nohadani, O., and Teo, K. M. Nonconvex robust optimization for problems with constraints. *INFORMS journal on Computing*, 22(1):44–58, 2010.

Bogunovic, I., Scarlett, J., Jegelka, S., and Cevher, V. Adversarially robust optimization with Gaussian processes. In *Conf. Neur. Inf. Proc. Sys. (NeurIPS)*, 2018.

Bogunovic, I., Krause, A., and Scarlett, J. Corruption-tolerant Gaussian process bandit optimization. In *Int. Conf. Art. Intel. Stats. (AISTATS)*, 2020a.

Bogunovic, I., Losalka, A., Krause, A., and Scarlett, J. Stochastic linear bandits robust to adversarial attacks. https://arxiv.org/abs/2007.03285, 2020b.

Bull, A. D. Convergence rates of efficient global optimization algorithms. *J. Mach. Learn. Res.*, 12(Oct.):2879–2904, 2011.

Cai, X. and Scarlett, J. On lower bounds for standard and robust Gaussian process bandit optimization. In *Proceedings of Machine Learning Research (PMLR)*, 2020. https://arxiv.org/abs/2008.08757.

Cai, X., Gomes, S., and Scarlett, J. Lenient regret and good-action identification in gaussian process bandits. In *Int. Conf. Mach. Learn. (ICML)*, 2021.

Chowdhury, S. R. and Gopalan, A. On kernelized multi-armed bandits. In *Int. Conf. Mach. Learn. (ICML)*, 2017.

Contal, E., Buffoni, D., Robicquet, A., and Vayatis, N. Parallel gaussian process optimization with upper confidence bound and pure exploration. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 225–240. Springer, 2013.

Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., and Leyton-Brown, K. Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In *NIPS Workshop on Bayesian Optimization in Theory and Practice*, 2013.

Garcelon, E., Roziere, B., Meunier, L., Teytaud, O., Lazaric, A., and Pirotta, M. Adversarial attacks on linear contextual bandits. In *Conf. Neur. Inf. Proc. Sys. (NeurIPS)*, 2020.

GPy. GPy: A Gaussian process framework in python. http://github.com/SheffieldML/GPy, 2012.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

Jaquier, N., Rozo, L., Calinon, S., and Bürger, M. Bayesian Optimization meets Riemannian Manifolds in Robot Learning. In *Conference on Robot Learning*, pp. 233–246. PMLR, 2020.

Jun, K.-S., Li, L., Ma, Y., and Zhu, J. Adversarial attacks on stochastic bandits. In *Conf. Neur. Inf. Proc. Sys. (NeurIPS)*, 2018.

Li, Z. and Scarlett, J. Gaussian process bandit optimization with few batches. https://arxiv.org/abs/2110.07788, 2021.

Lykouris, T., Mirrokni, V., and Paes Leme, R. Stochastic bandits robust to adversarial corruptions. In *ACM Symp. Theory Comp. (STOC)*, 2018.

Martinez-Cantin, R., Tee, K., and McCourt, M. Practical Bayesian optimization in the presence of outliers. In *Int. Conf. Art. Intel. Stats. (AISTATS)*, 2018.

Nogueira, J., Martinez-Cantin, R., Bernardino, A., and Jamone, L. Unscented Bayesian optimization for safe robot grasping. In *IEEE/RSJ Int. Conf. Intel. Robots and Systems (IROS)*, 2016.

Rasmussen, C. E. Gaussian processes for machine learning. MIT Press, 2006.

Rolland, P., Scarlett, J., Bogunovic, I., and Cevher, V. High-dimensional Bayesian optimization via additive models with overlapping groups. In *Int. Conf. Art. Intel. Stats. (AISTATS)*, pp. 298–307, 2018.

Snoek, J., Larochelle, H., and Adams, R. P. Practical Bayesian optimization of machine learning algorithms. In *Conf. Neur. Inf. Proc. Sys. (NIPS)*, pp. 2951–2959, 2012.

Srinivas, N., Krause, A., Kakade, S. M., and Seeger, M. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Int. Conf. Mach. Learn. (ICML)*, 2010.

Swersky, K., Snoek, J., and Adams, R. P. Multi-task Bayesian optimization. In *Conf. Neur. Inf. Proc. Sys. (NIPS)*, pp. 2004–2012, 2013.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014. URL http://arxiv.org/abs/1312.6199.

Vakili, S., Khezeli, K., and Picheny, V. On information gain and regret bounds in gaussian process bandits. https://arxiv.org/abs/2009.06966, 2020.

Vanchinathan, H. P., Nikolic, I., De Bona, F., and Krause, A. Explore-exploit in top-n recommender systems via gaussian processes. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, pp. 225–232, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450326681. doi: 10.1145/2645710.2645733. URL https://doi.org/10.1145/2645710.2645733.

Wang, Z. and Jegelka, S. Max-value entropy search for efficient Bayesian optimization. In *Int. Conf. Mach. Learn. (ICML)*, pp. 3627–3635, 2017.

Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., Murching, S., Nykodym, T., Ogilvie, P., Parkhe, M., et al. Accelerating the Machine Learning Lifecycle with MLflow. *IEEE Data Eng. Bull.*, 41(4):39–45, 2018.

A. Additional Experimental Details/Results

A.1. Implementation Details

Our implementation is based on Python 3.8, using Conda (Anaconda, 2016) and MLflow (Zaharia et al., 2018) to manage environments and experiments across multiple machines. Our implementation uses several standard machine learning and scientific packages, such as GPy (GPy, 2012), NumPy (Harris et al., 2020) and others. The various libraries and the specific versions used can be found in our supplied code, in the Conda environment file attack_bo.yml.

Implementations of the synthetic functions are from HPOlib2 (Eggensperger et al., 2013). We use the original authors' implementation (Wang & Jegelka, 2017) for the robot pushing objective functions (Robot3D and Robot4D).

A.2. Synthetic Functions

For convenience, we explicitly append the dimensionality to the name of the function. Firstly, we consider the 1D function

$$f(x) = (6x - 2)^2 \sin(12x - 4), \tag{16}$$

which we refer to as Synthetic1D. Then, we compare the attacks on commonly used optimization synthetic function benchmarks (Eggensperger et al., 2013): Forrester1D, Levy1D, Levy-Hard1D, Bohachevsky2D, Bohachevsky-Hard2D, Branin2D, Camelback2D, Hartmann6D. We have chosen these benchmarks so that we attack under a diverse range of conditions. The 1D and 2D functions are illustrated in Fig. 9. Levy-Hard1D and Bohachevsky-Hard2D are variants of Levy1D and Bohachevsky2D, but with different $\mathcal{R}_{\text{target}}$, chosen to increase the difficultly of the attack.

A.3. Time Horizon, Kernel, and Optimization Algorithm

For objective functions with 1 or 2 dimensions, we run the experiments with 10 initial points and 100 iterations. Due to the added difficulty in higher dimensions, we use 50 initial points and 250 iterations for the other experiments.

We adopt the Matérn-$5/2$ kernel, and initialize the kernel parameters with values learned from data sampled from f prior to the optimization:

- For 1D and 2D functions, we randomly select 100 points from a multi-dimensional grid where each dimension is evenly spaced;
- For higher-dimensional functions, we use a total of 1000 points using a mix of strategies. We sample 500 points using the same grid strategy as for the 1D and 2D functions, as well as an additional 500 random points sampled from a uniform distribution and are not confined to a grid.

The kernel parameters are then fixed, and not updated throughout the attack. This amounts to an "idealized" setting for the player in which the kernel is well-known even in advance. In Sec. A.7, we additionally present results for the case that the kernel parameters are learned online.

As mentioned in the main body, we focus primarily on attacking a standard form of GP-UCB with $\beta_t = 0.5 \log(2t)$. However, in Sec. A.8, we additionally consider a variant of GP-UCB with enlarged confidence bounds designed for improved robustness (Bogunovic et al., 2020a).

A.4. GP-UCB Attack Experiments

Discussion of functions used. As seen in Fig. 9, each target region $\mathcal{R}_{\text{target}}$ is rectangular, and defined by its centroid and length; the choices for each experiment are given in Table 2.

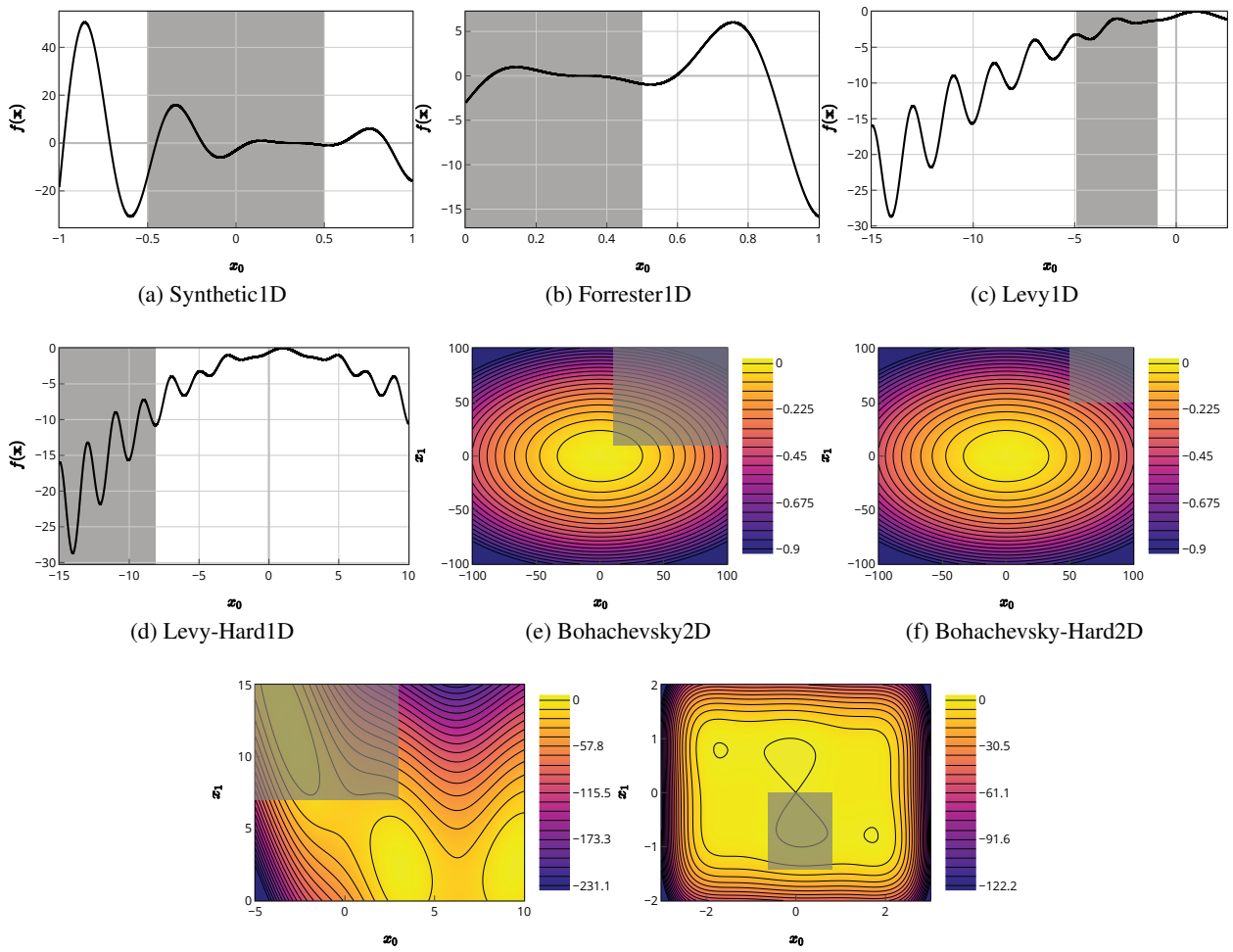
Synthetic1D and Forrester1D are relatively simple examples where the second-best peak falls inside of the $\mathcal{R}_{\text{target}}$ and there are few local maxima. Levy1D is an experiment with a periodic function, which increases the difficulty by the introduction of multiple local maxima. Levy-Hard1D uses the same function, but with a different domain and $\mathcal{R}_{\text{target}}$ such that the second-best peak falls outside of $\mathcal{R}_{\text{target}}$. Bohachevsky2D has a characteristic bowl shape with a single maximum, which is easy to optimize but difficult to attack. Here, we use a variant of the Bohachevsky2D function that is scaled by a constant factor to better manage the function range. Bohachevsky-Hard2D uses the same Bohachevsky2D function but with a smaller $\mathcal{R}_{\text{target}}$ to further increase the difficulty. Specifically, the function increases throughout $\mathcal{R}_{\text{target}}$ and when leaving $\mathcal{R}_{\text{target}}$, causing the attack to be especially difficult as explained in Sec. 3.3.

Branin2D has 3 global maxima, and $\mathcal{R}_{\text{target}}$ contains one of the global maxima. Camelback2D has 2 global maxima and several local maxima, and $\mathcal{R}_{\text{target}}$ contains one of the global maxima. As for an example with higher dimensionality, Hartmann6D is a 6-dimensional function with a global maximum lying outside $\mathcal{R}_{\text{target}}$.

Finally, we attack the 3-dimensional and 4-dimensional functions – Robot3D and Robot4D respectively (Wang & Jegelka, 2017). In both of these experiments, $\mathcal{R}_{\text{target}}$ is again rectangular, but with different lengths in each dimension.

Discussion of hyperparameters used. We consider 30 hyperparameter configurations (e.g., choices of Δ or $h_{\max}$) for each attack, choosing the relevant minimum and maximum values so that we sufficiently cover the entire spectrum of the behavior, e.g., configurations with low to high success rates. From Fig. 11-15, we can observe that higher costs may not associate with a higher success rate; this is also discussed in the main body.

In general, increasing a hyperparameter value is representative of increasing the aggressiveness of the attack. Here, we see examples of poorly chosen parameters (e.g., smallest hyperparameter value) having high cost yet attacking unsuccessfully. For example, from Fig. 12a and illustrated in Fig. 10 (top), we see that the poorly chosen hyperparameter $\Delta = 0$ for Clipping incurs a high cost with a poor success rate. In this case, the attack is *not aggressive enough* (i.e. 'under-aggressive'), and the GP-UCB algorithm is not incentivized to explore $\mathcal{R}_{\text{target}}$. Over many iterations, GP-UCB repeatedly explores $\mathcal{R}_{\text{target}}$, and continues to determine that $\mathcal{R}_{\text{target}}$ is no better than $\left(\tilde{\mathbf{x}}^*, f\left(\tilde{\mathbf{x}}^*\right)\right)$. Hence, the attack can incur a high cost due to GP-UCB continuing to choose points outside $\mathcal{R}_{\text{target}}$ across the entire time horizon.



(a) Synthetic1D (b) Forrester1D (c) Levy1D

(d) Levy-Hard1D (e) Bohachevsky2D (f) Bohachevsky-Hard2D

(g) Branin2D (h) Camelback2D

Figure 9: Illustrations for 1D and 2D experiments; $\mathcal{R}_{\text{target}}$ is shaded in each plot.

At the other end of the spectrum, increasing the aggressiveness of the attack can sometimes incur significantly more cost without any insignificant increase in success rate. For instance, we observe in Fig. 12a (see also Fig. 10) that the hyperparameter $\Delta = 42.5$ (bottom) incurs more cost then the most efficient $\Delta_{\text{efficient}} = 17.8$ (middle) but without a significant increase in success rate. Thus, the attack is overly aggressive; more cost than necessary is being used to incentivize the algorithm to choose actions in $\mathcal{R}_{\text{target}}$. In this particular case (Fig. 10), increasing Δ more than $\Delta_{\text{efficient}}$ increases the cost, as it unnecessarily, "swallows" more of the the local maxima on the right of $\mathcal{R}_{\text{target}}$. On the other hand, sometimes increasing the aggressiveness of the attack can have a minimal impact on the cost-success trade-off. For instance, we see this behavior in Fig. 11j, and Fig. 12h, where increasing aggression improves the success rate.

Considering that successfully forcing actions in $\mathcal{R}_{\text{target}}$ is the attacker's primary goal and that $\theta_{\text{efficient}}$ is difficult to find, in general, it may be preferable to be 'over-aggressive' than 'under-aggressive' when considering hyperparameters.

Comparison of attacks. In Fig. 16-17, we again see Clipping attacking efficiently and successfully, as discussed in Sec. 5. Similarly, we continue to observe that Subtraction Rnd and Subtraction Sq are often the next best attacks following Clipping. In Fig. 13-15, Subtraction Rnd has a slight advantage over Subtraction Sq, though the two are similar. Due to Subtraction Rnd's smooth $h(x)$, the attack behavior varies less with respect to its hyperparameter $h_{\max}$, and the attacks generally succeed with slightly less cost compared with Subtraction Sq.

The unique behavior of the various attacks in Bohachevsky2D is similar to the behavior observed for Bohachevsky-Hard2D. The efficiency of the Aggressive Subtraction attack in Bohachevsky-Hard2D continues to be similar to Bohachevsky2D, despite having a smaller $\mathcal{R}_{\text{target}}$ in Bohachevsky-Hard2D. Both Bohachevsky2D and Bohachevsky-Hard2D serve as illustrations of the difficult case discussed in Sec. 3.3.

	Name	Dim.	#Exps.
GP-UCB Attack	Synthetic1D	1	1510
	Forrester1D	1	1510
	Levy1D	1	1510
	Levy-Hard1D	1	1510
	Bohachevsky2D	2	1510
	Bohachevsky-Hard2D	2	1510
	Branin2D	2	1510
	Camelback2D	2	1510
	Hartmann6D	6	910
	Robot3D	3	910
	Robot4D	4	910
Kernels	Branin2D-RBF	2	1510
	Branin2D-Matérn $3/2$	2	1510
	Camelback2D-RBF	2	1510
	Camelback2D-Matérn $3/2$	2	1510
MaxVar	MaxVar-Forrester1D	1	1510
	MaxVar-Camelback2D	2	1510
	MaxVar-Robot3D	3	910
Online	Synthetic1D-Online	1	1510
	Forrester1D-Online	1	1510
	Bohachevsky2D-Online	2	1510
Defense	Levy1D $(C = 0.5, \eta^2 = 0.01)$	1	1510
	Levy1D $(C = 0.5, \eta^2 = 0.1)$	1	1510
	Levy1D $(C = 0.5, \eta^2 = 1)$	1	1510
	Levy1D $(C = 2, \eta^2 = 0.01)$	1	1510
	Levy1D $(C = 2, \eta^2 = 0.1)$	1	1510
	Levy1D $(C = 2, \eta^2 = 1)$	1	1510
	Levy1D $(C = 8, \eta^2 = 0.01)$	1	1510
	Levy1D $(C = 8, \eta^2 = 0.1)$	1	1510
	Levy1D $(C = 8, \eta^2 = 1)$	1	1510
Dynamic	Synthetic1D-Dynamic	1	500
	Forrester1D-Dynamic	1	500
	Levy1D-Dynamic	1	500

Table 1: Summary of various experiments and functions used, sorted by type and dimensionality.

Name	Centroid	Length
Synthetic1D	(0)	1
Forrester1D	(0.25)	0.5
Levy1D	(-2.915)	4
Levy-Hard1D	(-11.56)	6.881
Bohachevsky2D	$(55, 55)$	90
Bohachevsky-Hard2D	$(75, 75)$	50
Branin2D	$(-1, 11)$	8
Camelback2D	$(0.090, -0.713)$	1.425
Robot3D	$(2.5, 2.5, 20)$	$(5, 5, 20)$
Robot4D	$(2.5, 2.5, 20, \pi/2)$	$(5, 5, 20, \pi)$
Hartmann6D	$(0.6, \cdots, 0.6)$	0.8

Table 2: Summary of $\mathcal{R}_{\text{target}}$ parameters.

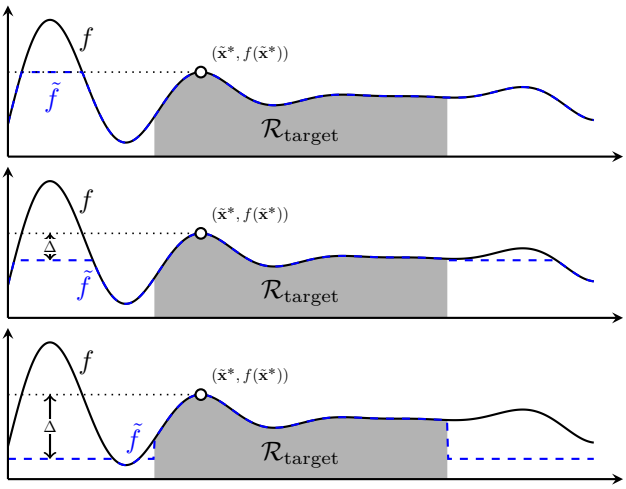


Figure 10: Clipping attack on Synthetic1D with 'under-aggressive' $\Delta = 0$ (top), most efficient $\Delta_{\text{efficient}} = 17.8$ (middle), and 'over-aggressive' $\Delta = 42.5$ (bottom). See (15) for the definition of "most efficient".

A.5. Kernel Experiments

We have focused on the Matérn $5/2$ kernel, as it is widely-adopted and provides useful properties that we can exploit in our theoretical analysis. Here we additionally demonstrate our attacks on two other popular kernel functions: Radial Basis Function (RBF) and Matérn $3/2$. Apart from the kernel choice, the setup is the same as that of Sec. A.4.

From Fig. 18, we observe that the RBF kernel is easier to attack as it is smoother than Matérn $5/2$; the RBF kernel corresponds to a Matérn kernel where $\nu \to \infty$, leading to infinitely many derivatives. Thm. 1 indicates that smoother functions (smaller γ_t) are easier to attack. Similarly, Matérn $3/2$ is harder to attack, which is consistent with it having a larger γ_t. Intuitively, using a less smooth kernel encourages more exploration, which makes it more difficult to force the algorithm into the target region.

A.6. MaxVar + Elimination Attack Experiments

Here we move from attacking GP-UCB to attacking the MaxVar+Elimination algorithm, described in Sec. 3.1. With the exception of the selection rule itself, other settings in each of the MaxVar attack experiments are identical to the corresponding GP-UCB attack experiments.

We observe from Fig. 19 that the behavior of both MaxVar-Camelback2D and MaxVar-Robot3D are similar to the GP-UCB counterparts. Notwithstanding the non-standard 'gap' as observed in the figure, MaxVar-Forrester1D behaved similarly when compared to Forrester1D. One exception is the 'gap' in Fig. 19a, which may be due to the 'non-smooth' selection criteria of the t-th point, explicitly filtering candidate points that are not as high as the highest LCB.

A.7. Online-Learned Kernel Experiments

In GP-UCB Attack, we initialized the kernel parameters with values learned from data sampled from f prior to the optimization. Here, we consider an alternative setting in which the kernel parameters are learned online. The kernel parameters here are updated using the widely-used technique – maximum likelihood optimized with the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm. Our experiments use the same settings as the corresponding GP-UCB attack experiments, with the exception of the kernel parameters.

Comparing the online vs. offline approaches in Fig. 20, we see that there is generally no major difference in the behavior of these two approaches; being online vs. offline does not appear to significantly impact robustness. A possible exception is that some *specific runs* (i.e., random seeds) appear to have an unusually low success rate in Synthetic1D-Online when online kernel learning is used, particularly with Subtraction Rnd. A possible explanation for this is that the attack can sometimes perturb the function to the extent that the maximum-likelihood kernel parameters are highly misleading, causing the algorithm to behave in an unpredictable manner, and choose fewer points from $\mathcal{R}_{\text{target}}$.

A.8. Defense Experiments

We consider defending against our attacks using a combination of two techniques. Firstly, we use a technique proposed in (Bogunovic et al., 2020a), namely, adding a constant C to the exploration parameter:

$$\beta_t = 0.5 \log(2t) + C. \tag{17}$$

Secondly, we increase the model's robustness to adversarial noise via the model's noise variance η^2 in the posterior update equations:

$$\begin{aligned} \mu_{t+1}(\mathbf{x}) &= \mathbf{k}_t(\mathbf{x})^\top \left(\mathbf{K}_t + \eta^2 \mathbf{I}_t\right)^{-1} \mathbf{y}_t, \\ \sigma_{t+1}(\mathbf{x}) &= k(\mathbf{x},\mathbf{x}) - \mathbf{k}_t(\mathbf{x})^\top \left(\mathbf{K}_t + \eta^2 \mathbf{I}_t\right)^{-1} \mathbf{k}_t(\mathbf{x}). \end{aligned} \tag{18}$$

We vary $C \in \{0.5, 2, 8\}$ with $\eta^2 \in \{0.01, 0.1, 1\}$, resulting in nine different parameter combinations, allowing us to study their combined effect.

We observe from Fig. 21 that increasing C indeed decreases the attack success rate and increases the cost for each run, though seemingly not in a very major way. Essentially, increasing the confidence width causes GP-UCB to become more explorative and less exploitative, making it harder for the attacker to quickly steer the algorithm towards $\mathcal{R}_{\text{target}}$, and hence increasing the attack cost.

Similarly, we see that increasing η^2 tends to increase the cost for each run. By increasing η^2, we make the model expect more noise, and thus be more cautious against 'unusual' observations. Thus, the adversary needs to spend more cost to perform the attack. This defense works significantly better on the Subtraction Attacks (Subtraction Rnd, Subtraction Sq) than on Clipping and Aggressive Subtraction. We believe that this is because Clipping and Aggressive Subtraction perturb the function across the majority of the domain rather than a few localized regions. The latter can potentially be written off as random noise more easily than the former.

The higher the value of C used in (17) and of η^2 used in (18), the more similar the attacks tend to behave, and the less sensitive the algorithm tends to be to varying the attacker's hyperparameters. We observe from Fig. 21i that the combination of both simple defense techniques on Subtraction Rnd and Subtraction Sq increase robustness.

Overall, we believe that this experiment, and our work in general, motivates the study of further defenses beyond the simple defenses proposed, particularly when targeting applications in which robustness is critical.

A.9. Dynamic Experiments

We have focused on the fixed-hyperparameter setting, where the hyperparameters are fixed prior to the attack (e.g., Δ is predetermined and fixed in the Aggressive Subtraction Attack). This setting serves as an important stepping stone to more general settings, and also aligns with our theory.

On the other hand, from our experiments and discussion, it is evident that the choice of the attack's hyperparameters can be a very important factor in the success of the attack. If f is known

to the attacker, then in principle various configurations could be simulated to find the best one. However, pre-specifying the hyperparameters may be much more difficult when limited prior knowledge is available.

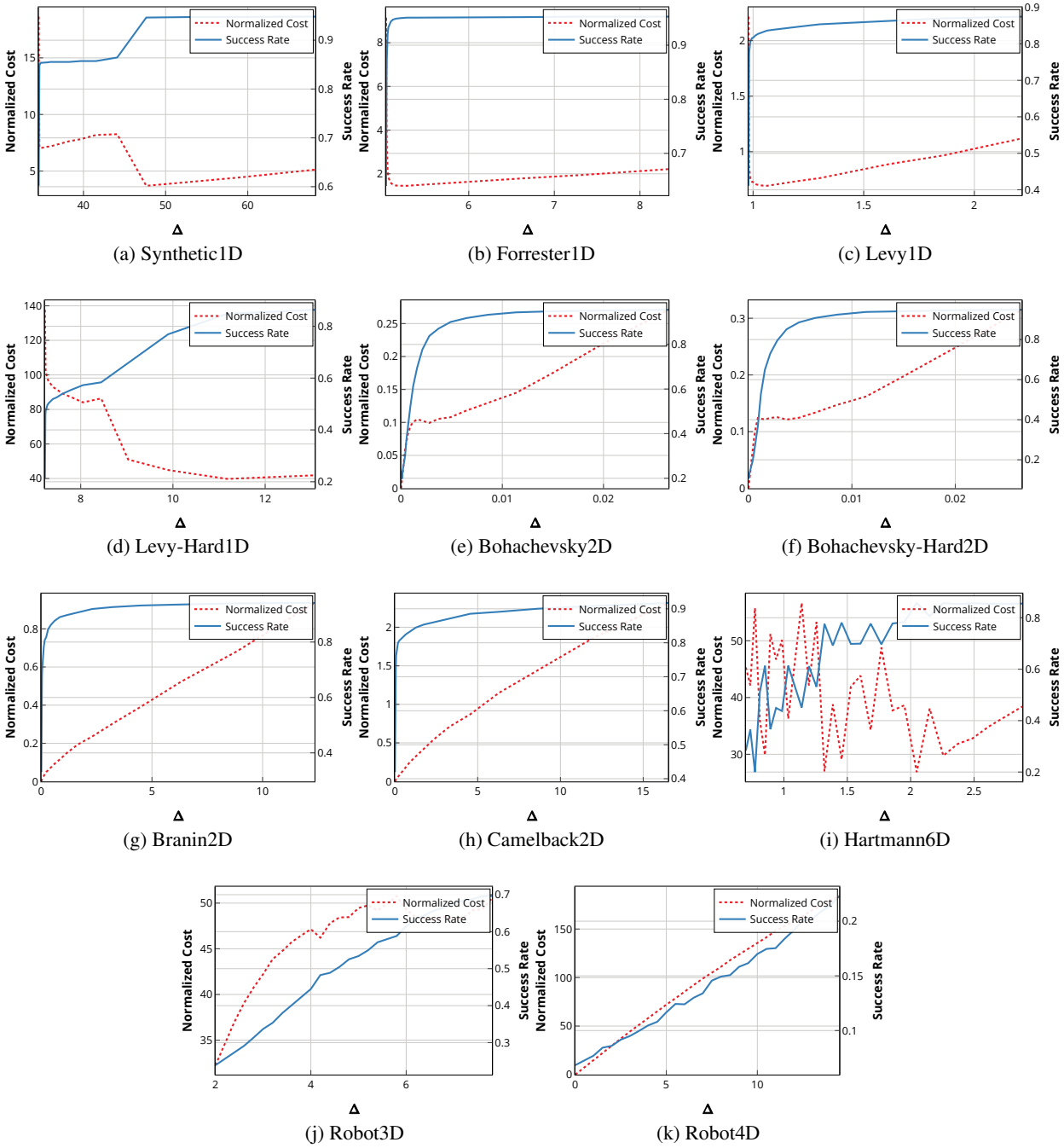
Here, we provide an initial investigation into whether the hyperparameters can be chosen automatically, i.e., dynamically adjusted online during the attack. We consider a simple strategy to adjust the hyperparameter: The attack gets less aggressive whenever $\mathcal{R}_{\mathrm{target}}$ is sampled consecutively K times, and more aggressive otherwise. Thus, the hyperparameter (e.g., representing Δ) is updated as follows:

$$\theta_t = \begin{cases} \theta_{t-1} - F \cdot \theta_{t-1} & \text{sampled consecutively,} \\ \theta_{t-1} + F \cdot \theta_{t-1} & \text{otherwise,} \end{cases} \tag{19}$$

where F is the size of the hyperparameter update, as a fraction of the current value.

Apart from this change, we consider the same setup as that of Sec. A.4. We apply the strategy to Aggressive Subtraction on Synthetic1D, Forrester1D and Levy1D; based on minimal manual tuning, we set $F = 0.1$ and $K = 3$ across the three functions without further tweaking. We compare the fixed vs. dynamic attacks for various hyperparameters, where setting a hyperparameter in the dynamic setting means fixing its *initial* value only.

From Fig. 22, this strategy works well on these functions, significantly improving the performance (i.e., trade-off of success rate and cost) for the configurations that perform poorly in the case of being fixed. This experiment serves as a useful proof of concept, but further research is needed towards establishing fully automated attacks method for general scenarios. We believe that our work provides a good starting point towards this goal.



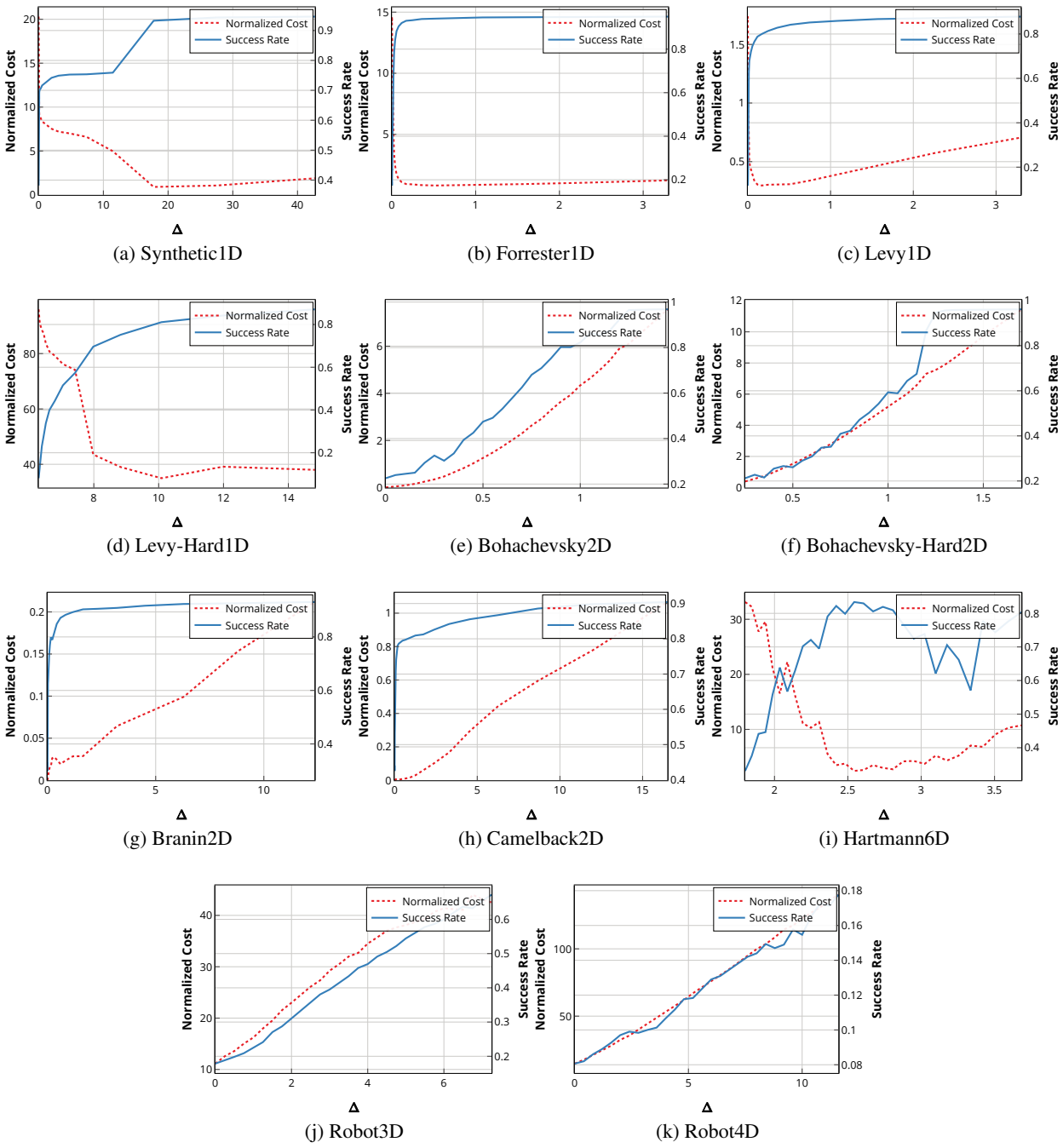
(a) Synthetic1D (b) Forrester1D (c) Levy1D

(d) Levy-Hard1D (e) Bohachevsky2D (f) Bohachevsky-Hard2D

(g) Branin2D (h) Camelback2D (i) Hartmann6D

(j) Robot3D (k) Robot4D

Figure 11: Effect of Aggressive Subtraction's hyperparameter Δ on the success rate and cost incurred.



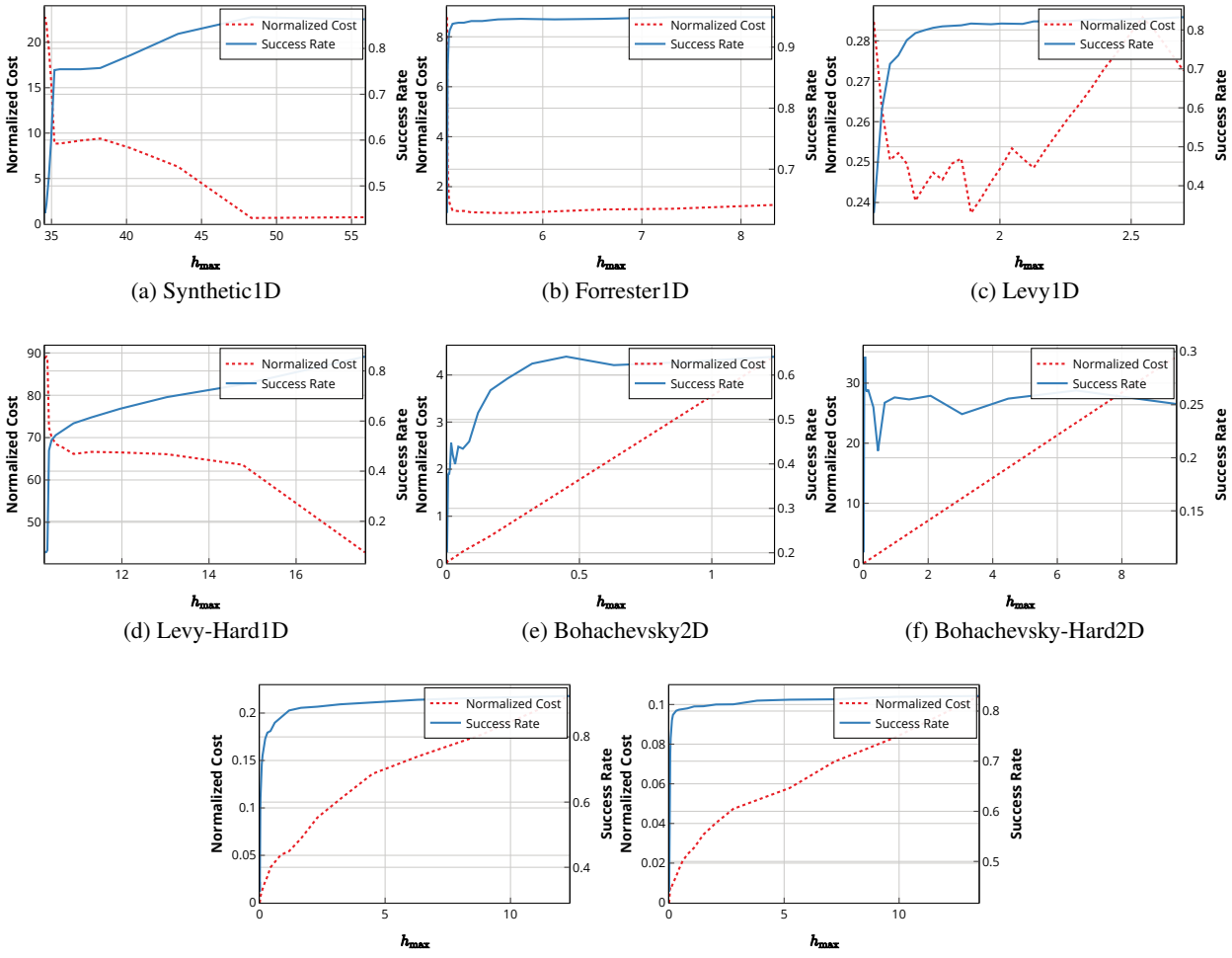
(a) Synthetic1D (b) Forrester1D (c) Levy1D

(d) Levy-Hard1D (e) Bohachevsky2D (f) Bohachevsky-Hard2D

(g) Branin2D (h) Camelback2D (i) Hartmann6D

(j) Robot3D (k) Robot4D

Figure 12: Effect of Clipping's hyperparameter Δ on the success rate and cost incurred.

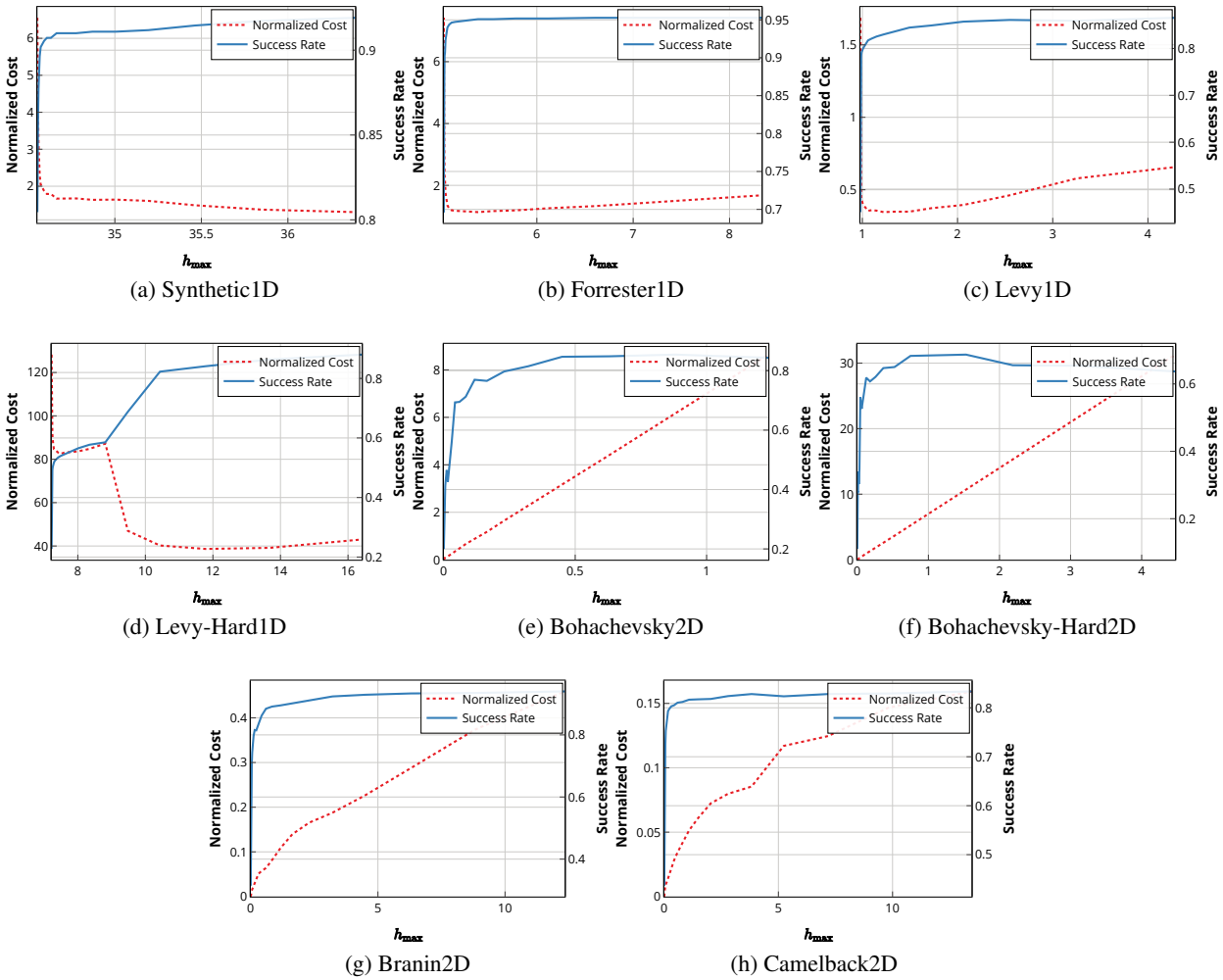


(a) Synthetic1D (b) Forrester1D (c) Levy1D

(d) Levy-Hard1D (e) Bohachevsky2D (f) Bohachevsky-Hard2D

(g) Branin2D (h) Camelback2D

Figure 13: Effect of Subtraction Rnd's hyperparameter $h_{\max}$ on the success rate and cost incurred.



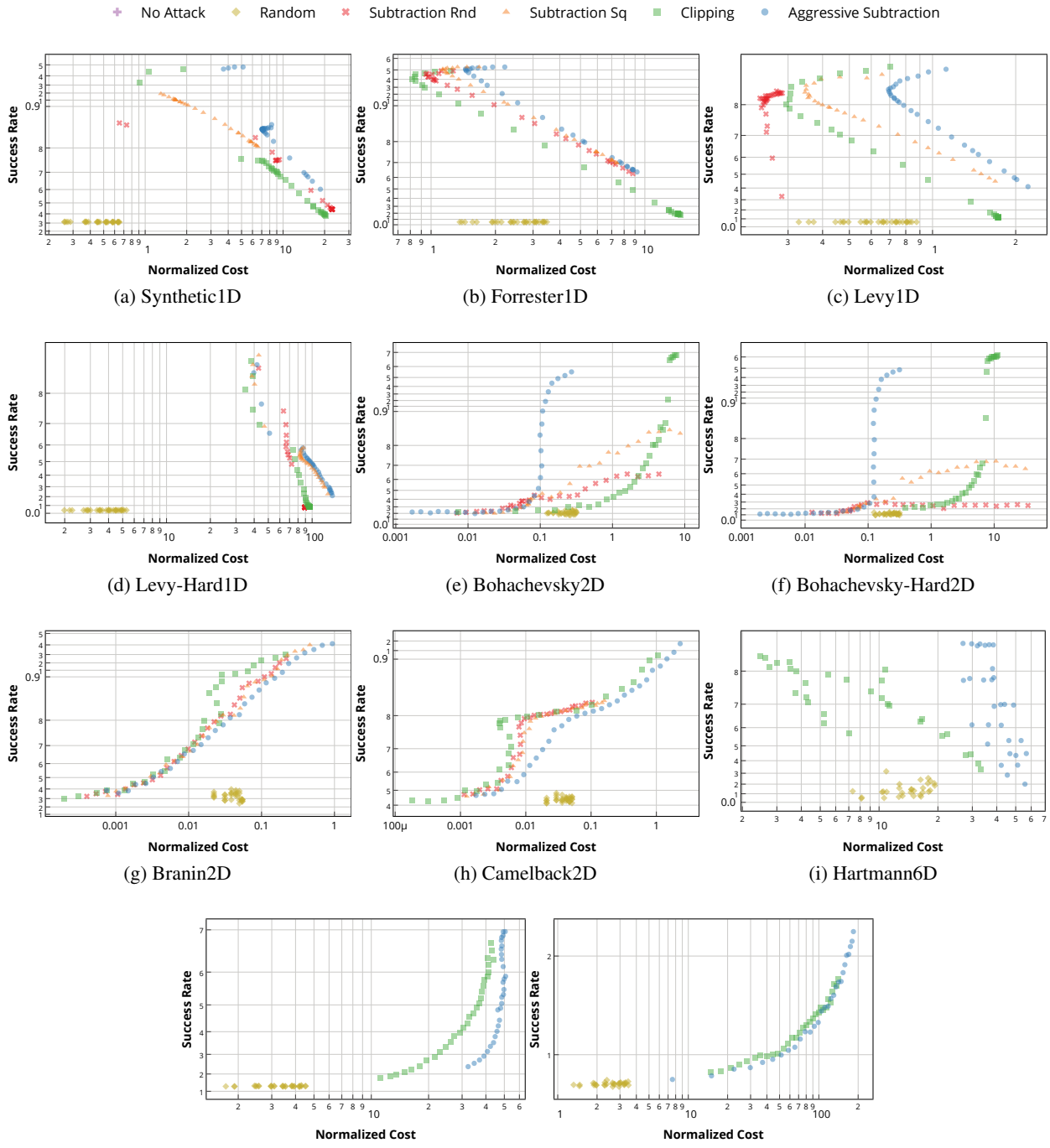
(a) Synthetic1D (b) Forrester1D (c) Levy1D

(d) Levy-Hard1D (e) Bohachevsky2D (f) Bohachevsky-Hard2D

(g) Branin2D (h) Camelback2D

Figure 14: Subtraction Sq

Figure 15: Effect of Subtraction Sq's hyperparameter $h_{\max}$ on the success rate and cost incurred.



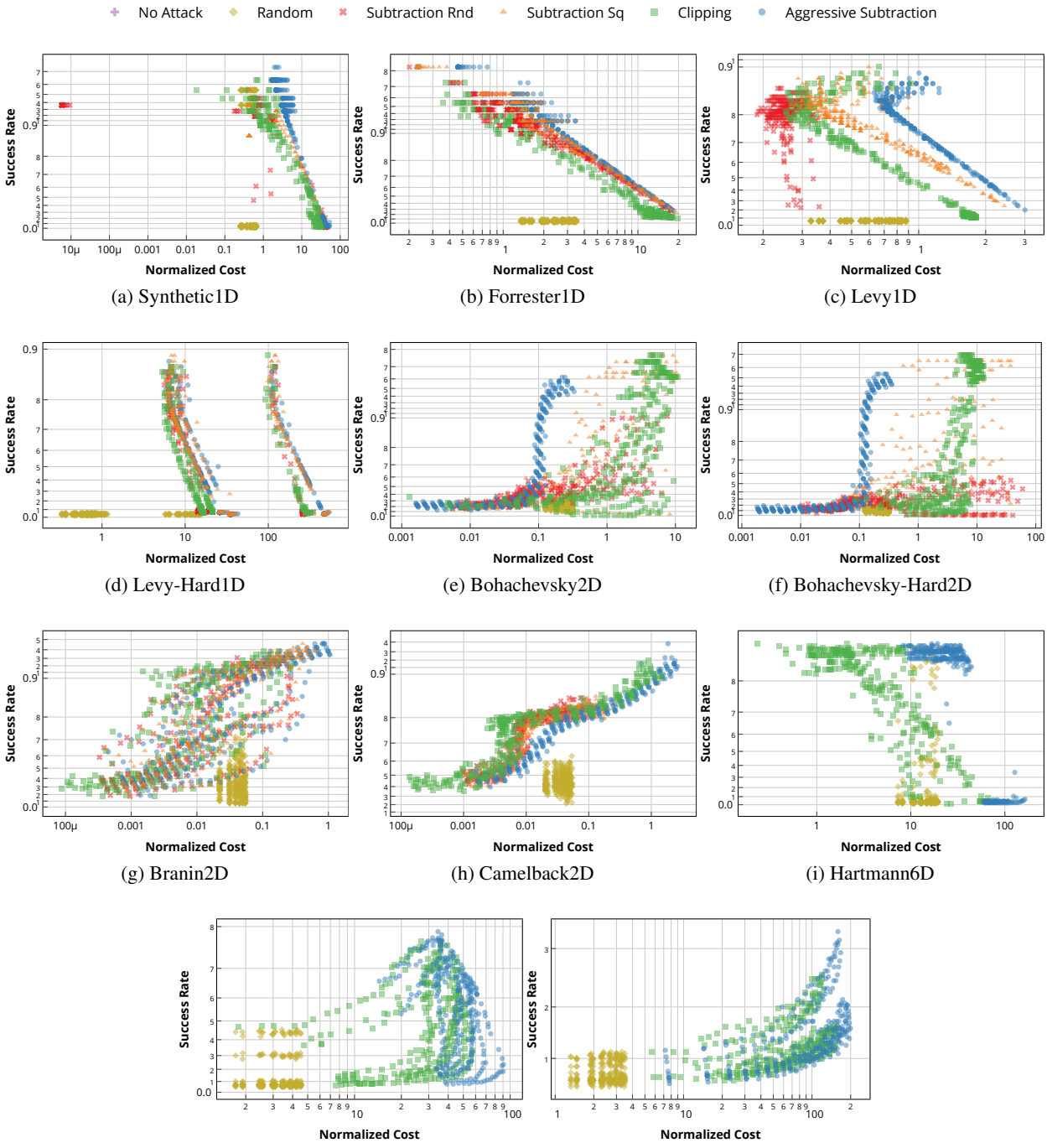
(a) Synthetic1D (b) Forrester1D (c) Levy1D

(d) Levy-Hard1D (e) Bohachevsky2D (f) Bohachevsky-Hard2D

(g) Branin2D (h) Camelback2D (i) Hartmann6D

(j) Robot3D (k) Robot4D

Figure 16: Success rate vs. cost averaged over random seeds, each point is the performance of a particular hyperparameter.



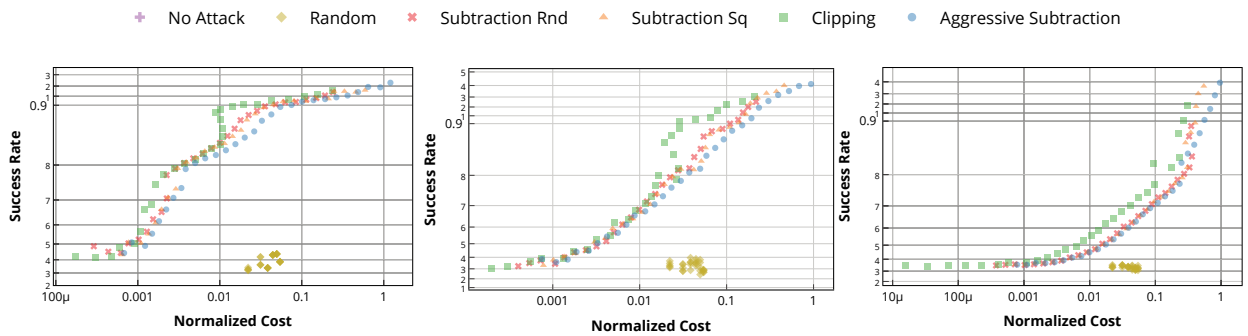
(a) Synthetic1D (b) Forrester1D (c) Levy1D

(d) Levy-Hard1D (e) Bohachevsky2D (f) Bohachevsky-Hard2D

(g) Branin2D (h) Camelback2D (i) Hartmann6D

(j) Robot3D (k) Robot4D

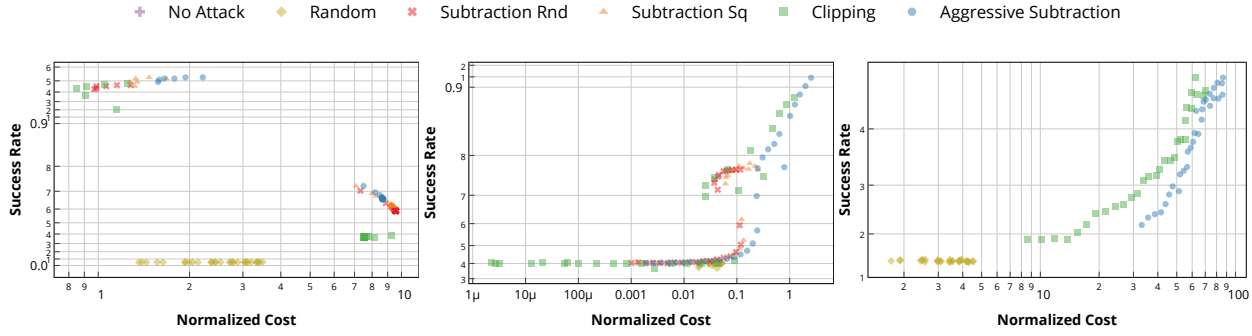
Figure 17: Success rate vs. cost with every random seed shown individually, i.e., each point is a single run.



(a) Branin2D-RBF (b) Branin2D-Matérn $5/2$ (c) Branin2D-Matérn $3/2$

(d) Camelback2D-RBF (e) Camelback2D-Matérn $5/2$ (f) Camelback2D-Matérn $3/2$

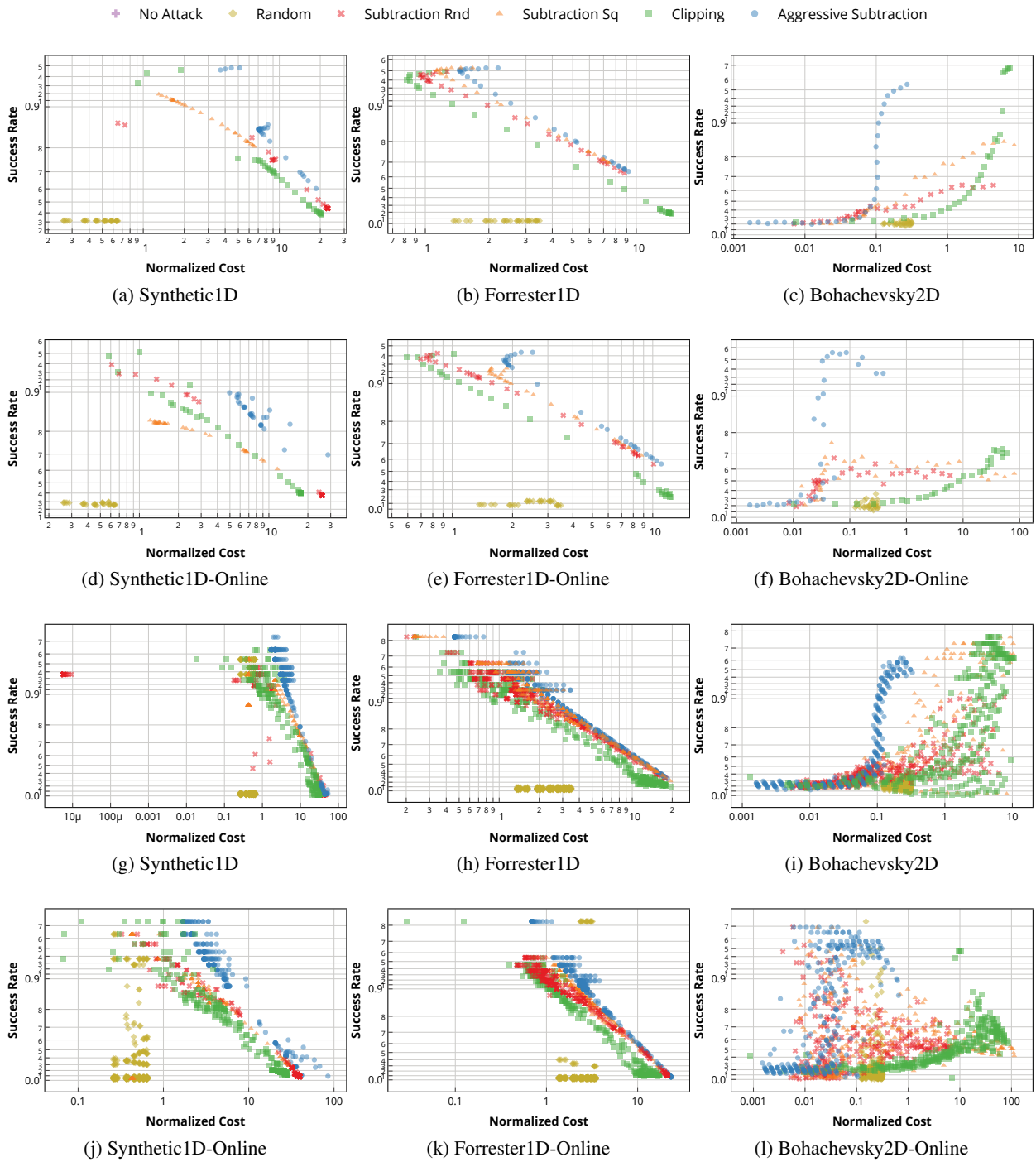
Figure 18: Experiments comparing different kernels: Radial Basis Function (RBF), Matérn $5/2$, and Matérn $3/2$.



(a) MaxVar-Forrester1D (b) MaxVar-Camelback2D (c) MaxVar-Robot3D

(d) MaxVar-Forrester1D (e) MaxVar-Camelback2D (f) MaxVar-Robot3D

Figure 19: We attack the MaxVar + Elimination algorithm in 3 experiments.



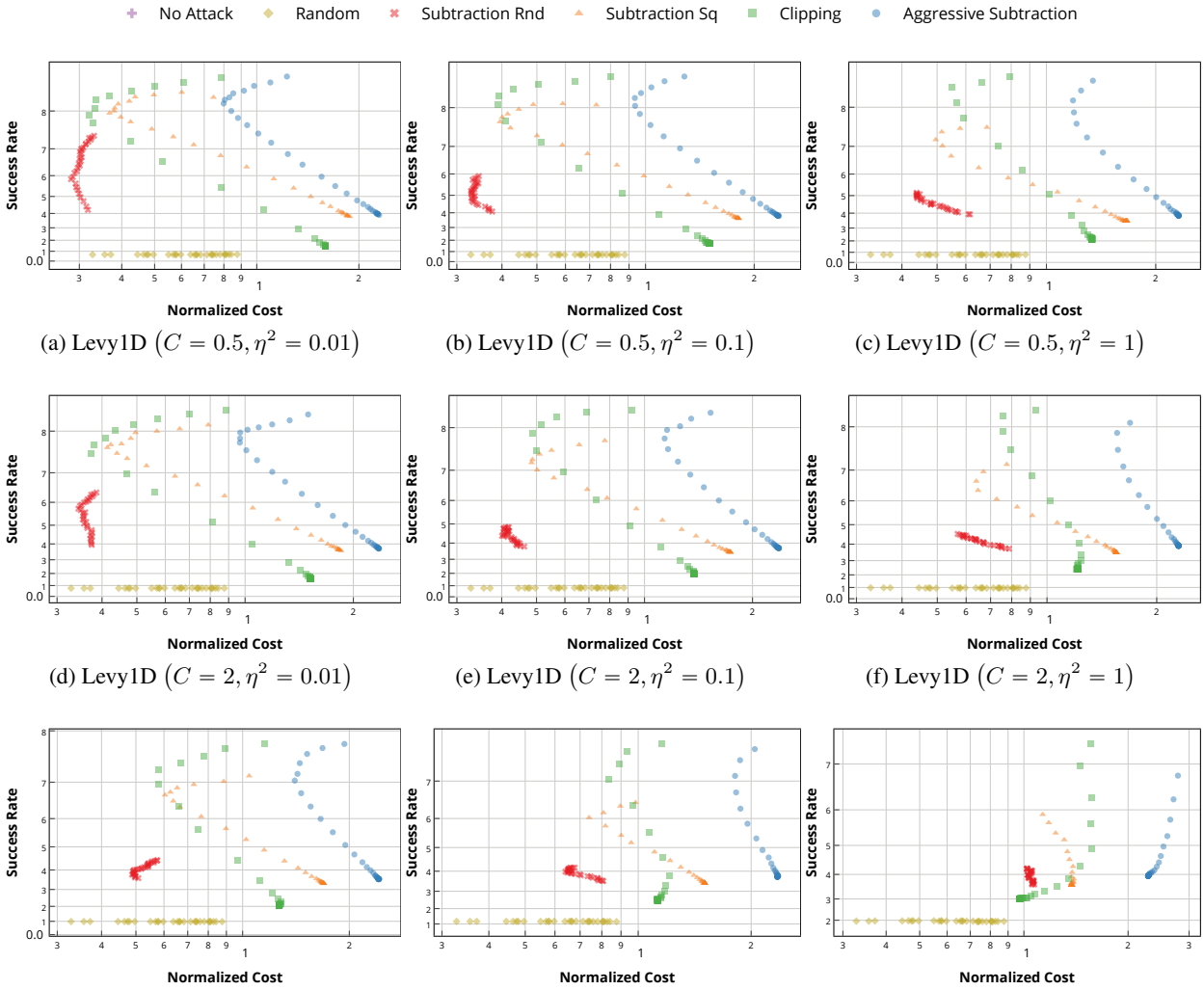
(a) Synthetic1D (b) Forrester1D (c) Bohachevsky2D

(d) Synthetic1D-Online (e) Forrester1D-Online (f) Bohachevsky2D-Online

(g) Synthetic1D (h) Forrester1D (i) Bohachevsky2D

(j) Synthetic1D-Online (k) Forrester1D-Online (l) Bohachevsky2D-Online

Figure 20: Experiments comparing the kernel parameters being learned online (second and forth rows) vs. learned from data sampled from f prior to the optimization (first and third rows). The top two rows average over random seeds, and the bottom two rows show every individual run.

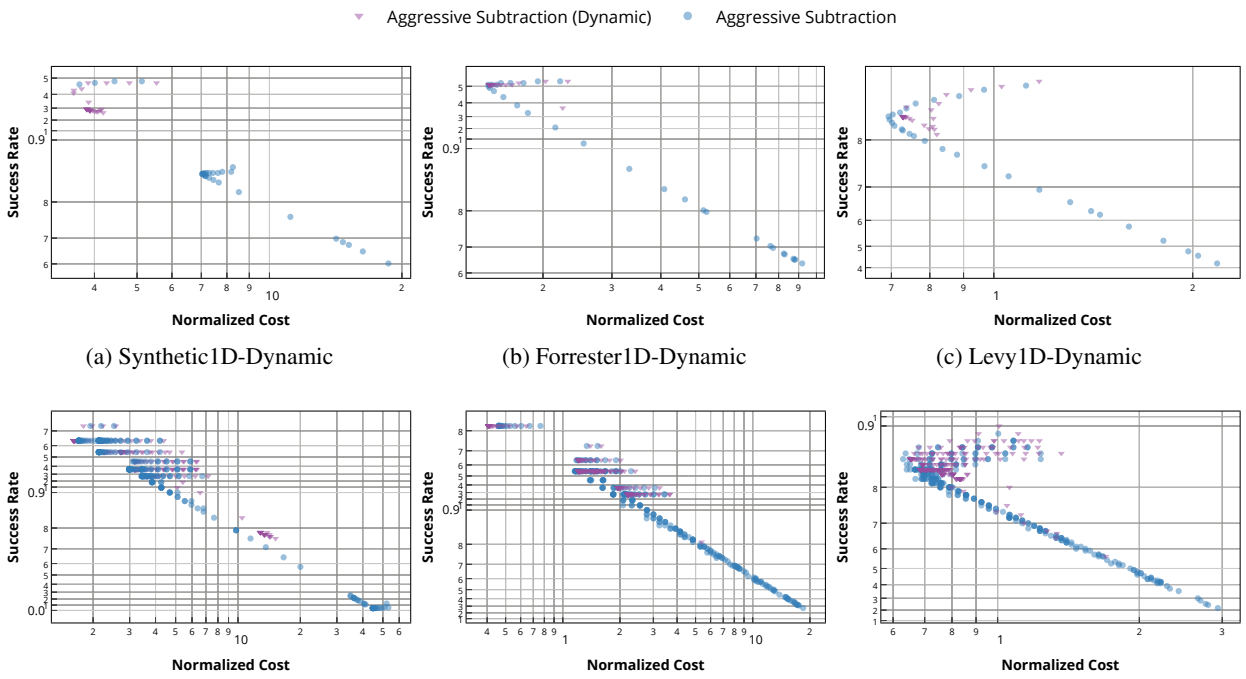


(a) Levy1D $\left(C = 0.5, \eta^2 = 0.01\right)$ (b) Levy1D $\left(C = 0.5, \eta^2 = 0.1\right)$ (c) Levy1D $\left(C = 0.5, \eta^2 = 1\right)$

(d) Levy1D $\left(C = 2, \eta^2 = 0.01\right)$ (e) Levy1D $\left(C = 2, \eta^2 = 0.1\right)$ (f) Levy1D $\left(C = 2, \eta^2 = 1\right)$

(g) Levy1D $\left(C = 8, \eta^2 = 0.01\right)$ (h) Levy1D $\left(C = 8, \eta^2 = 0.1\right)$ (i) Levy1D $\left(C = 8, \eta^2 = 1\right)$

Figure 21: Variants of GP-UCB which add a constant C to the confidence width in order to defend against the attack.



(a) Synthetic1D-Dynamic (b) Forrester1D-Dynamic (c) Levy1D-Dynamic

(d) Synthetic1D-Dynamic (e) Forrester1D-Dynamic (f) Levy1D-Dynamic

Figure 22: Experiments applying a simple dynamic hyperparameter strategy for the Aggressive Subtraction attack. The top row averages over random seeds, and the bottom row shows every individual run.