Are Evolutionary Algorithms Safe Optimizers?

Are Evolutionary Algorithms Safe Optimizers?

Youngmin Kim
youngmin.kim@manchester.ac.uk
Alliance Manchester Business School,
University of Manchester
Manchester, UK

Richard Allmendinger
richard.allmendinger@manchester.ac.uk
Alliance Manchester Business School,
University of Manchester
Manchester, UK

Manuel López-Ibáñez[*]
manuel.lopez-ibanez@manchester.ac.uk
Alliance Manchester Business School,
University of Manchester
Manchester, UK

ABSTRACT

We consider a type of constrained optimization problem, where the violation of a constraint leads to an irrevocable loss, such as breakage of a valuable experimental resource/platform or loss of human life. Such problems are referred to as safe optimization problems (SafeOPs). While SafeOPs have received attention in the machine learning community in recent years, there was little interest in the evolutionary computation (EC) community despite some early attempts between 2009 and 2011. Moreover, there is a lack of acceptable guidelines on how to benchmark different algorithms for SafeOPs, an area where the EC community has significant experience in. Driven by the need for more efficient algorithms and benchmark guidelines for SafeOPs, the objective of this paper is to reignite the interest of this problem class in the EC community. To achieve this we (i) provide a formal definition of SafeOPs and contrast it to other types of optimization problems that the EC community is familiar with, (ii) investigate the impact of key SafeOP parameters on the performance of selected safe optimization algorithms, (iii) benchmark EC against state-of-the-art safe optimization algorithms from the machine learning community, and (iv) provide an open-source Python framework to replicate and extend our work.

CCS CONCEPTS

• **Mathematics of computing** → **Continuous optimization**; • **Theory of computation** → *Evolutionary algorithms*; • **Computing methodologies** → *Gaussian processes.*

KEYWORDS

safe optimization, safety constraints, constrained optimization, Bayesian optimization, benchmarking

1 INTRODUCTION

This work focuses on *safe optimization problems*, a special type of constrained optimization problem that has received rather little attention in the evolutionary computation (EC) community, but more so in the machine learning community. Such problems are subject to constraints that, when violated, result in an irrevocable loss of valuable experimental platform/resource, such as breakage of a machine used for experiments, or even injury to a patient [11]. Here, these constraints are referred to as safety constraints, and evaluations of input points (candidate solutions) that violate a safety constraint are called unsafe evaluations. Typically, the objective function(s) and safety constraint function(s) (these concepts will be explained in detail in Section 2) are given as black-box functions, and each evaluation is expensive. Examples of safe optimization problems (safeOPs) include clinical experiments [16, 17], controller optimization for quadrotor vehicle [4, 5, 8], engine calibration [10, 14], and simulation-based optimization [2, 13].

There are two algorithmic branches in safe optimization [11]: Safe optimization through evolutionary algorithms (EAs) vs Gaussian process (GP) regression (in the remainder of this paper, algorithms belonging to each branch will be denoted as safe EA and safe GP algorithm, respectively). Although safe optimization was first considered by the EC community in 2009 [10] and 2011 [1], we are not aware of any further research on this topic. In comparison, the machine learning community has actively worked on SafeOPs from 2015.

Research on safe optimization is fragmented with no unified guidelines on how to benchmark algorithms for safe optimization. When algorithms for safeOPs are benchmarked, an aspect of performance relates to the best objective function values achieved [1, 2, 5, 10, 13, 14, 16, 17]. Another aspect of performance relates to safety, which can be measured, for example, by the number of unsafe evaluations (or, equivalently, the number of safe evaluations) at each iteration or evaluation step [2, 5, 10, 13], the proportion of survived solutions, i.e., u_t/u_0 where u_0 is the initial parent population size and u_t is the number of survived offspring size at t^{th} iteration step (individuals violating the safety constraint are removed) [1], sensitivity and specificity on the evaluations [14], and the size of safe set (i.e., the number of input points inferred to be safe) [16, 17]. However, safeOP benchmarking has been conducted within a single domain only where an algorithm is proposed. In such studies, the capabilities of the proposed algorithms under slightly different safeOP scenarios are seldom examined.

The contributions made by our paper are as follows:

(1) We provide a formal definition of safeOPs assuming the real-world application, and contrast it to other types of problem that the EC community is familiar with.
(2) We investigate the impact on performance of safe optimization algorithms for several key parameters affecting the complexity of safeOPs.
(3) This is the first research that compares safe EA with safe GP algorithms. Previous studies looked at the two algorithm types in isolation.
(4) We propose an initial set of guidelines to carry out a benchmark study. This is accompanied with an open-source Python framework made available to the community to facilitate replication and extension of our work.

The remainder of the paper is organized as follows. In Section 2, a formal definition of safeOPs used for our experiments is presented.

[*]Also with School of Computer Science, University of Málaga.

Section 3 describes the working principles of existing safe optimization algorithms, and Section 4 provides the experimental setup for the benchmark study carried out in Section 5. Finally, conclusions and future research are discussed in Section 6.

2 PROBLEM STATEMENT

A safe optimization problem (SafeOP) can be formally defined as:

$$\begin{aligned} \text{maximize} \quad & y(\mathbf{x}) = f(\mathbf{x}) + \epsilon\,, \quad \epsilon \sim \mathcal{N}(0, \sigma^2), \;\; \mathbf{x} \in D \subset \mathbb{R}^d \\ \text{s.t.} \quad & y(\mathbf{x}) \geq h \end{aligned} \tag{1}$$

where $\mathbf{x} = (x_1, \dots, x_d)$ is a d-dimensional input point (candidate solution), D is the feasible search space (defined by standard feasibility constraints), $f : D \to \mathbb{R}$ is an objective function that is black-box and expensive to evaluate, and observed as a value $y(\mathbf{x}) = f(\mathbf{x}) + \epsilon$ perturbed by additive noise [5, 16]. For simplicity, here $y(\mathbf{x})$ is both the objective function and a *safety function* at the same time (i.e., evaluating an input point whose $y(\mathbf{x})$ value is less than or equal to h is unsafe), and h is a known *safety threshold* (a constant). Violating a safety constraint (i.e. evaluating a solution for which $y(\mathbf{x}) < h$ holds) results in an irrevocable scenario, such as loss of an expensive kit or injury of a patient (more example below). The *safety budget* determines the number of times (usually ranging from 0 to a small integer number) the safety constraint can be violated (e.g. in case a limited number of kits are available). The optimization process of a SafeOP is terminated either if the evaluation budget or the safety budget has been exhausted. To kickstart the optimization of a SafeOP, in practice the optimizer is often provided a set of known safe solutions (also known as the *initial safe seed*).

A real-world example of a SafeOP is spinal cord therapy where experiments were conducted to rats suffering spinal cord injuries [16]. The goal of the study was to search for electrical-stimulating configurations that optimize the resulting activity in lower limb muscles, to improve spinal reflex and locomotor function. Here, unsafe evaluations have negative impacts on rehabilitation, or even painful sometimes. This is an example of zero-budget scenario on unsafe evaluation.

The definition of a SafeOP can be made arbitrarily more complex without being less realistic, for example, by having an unknown safety threshold *h*, which itself can be a function of e.g. $\mathbf{x}$, or multiple black-box objective and safety constraint functions. However, for the sake of simplicity, here we limit ourselves to a single safety constraint with a known constant *h*.

Although SafeOPs share similarities with standard constrained optimization problems [7, 12], which are subject to feasibility constraints only, there are important differences. Feasibility constraints reflect some aspects relevant for an input point $\mathbf{x}$ to be of practical use, for instance, bounds of instrument settings, physical limitations, or design requirements [11]. Also, there is no limit on the number of violations of a feasibility constraint, and violation of a feasibility constraint can be predicted when mathematical models are given (meaning violations can be avoided). If an infeasible solution is evaluated, nothing severe happens to the optimizer or the environment. The objective function value of infeasible solutions is either undefined or simply penalized to encourage the optimizer to remain in the feasible search space. In comparison, violating a safety constraint beyond the safety budget results in premature termination of an optimization run, and predicting violations is more tricky because of their black-box nature. A solution $\mathbf{x}$ can be feasible but unsafe, and vice versa. Furthermore, in safe optimization, an optimization algorithm that achieves a worse performance in terms of f but violates a safety constraint less often may be preferred over an algorithm that violates a safety constraint more often to achieve a better performance in f. The cost of violating a safety constraint may be a key factor in balancing this performance-safety trade-off.

To keep things simple, in this work we assume no feasibility constraints (apart from box constraints on the decision variable values x_i).

3 SAFE OPTIMIZATION ALGORITHMS: A BRIEF REVIEW

EAs for safe optimization. SafeOPs were initially studied in the EC community between 2009 and 2011. A violation avoidance (VA) method [10] was proposed in 2009, as a universal approach that can be augmented onto any EA when faced with a SafeOP. VA replaces the offspring generation process of an EA by rejecting and then regenerating an offspring solution if its closest solution (or nearest neighbor) in the decision space out of all previously evaluated solutions (i.e. the search history) was unsafe. In [1], a reconfigurable, destructible and unreplaceable experimental platform in closed-loop optimization setting was considered using three different stochastic optimizers: Tournament selection based genetic algorithm (TGA), reproduction of best solutions (RBS), and a population of stochastic hill-climbers (PHC). The three optimizers do not have a functionality of inferring the safety of an input point before it is actually evaluated; however, they guide the population away from unsafe regions in the search space by prohibiting unsafe input points from entering the population.

GPs for safe optimization. SafeOPs have been actively studied in the machine learning community from 2015. SafeOpt [16] was proposed to deal with a single-objective safeOP subject to a single safety constraint inspiring other safe GP algorithms. Later, a modified SafeOpt [5] was proposed that makes weaker assumptions about the black-box objective function; SafeOpt-MC [4] was designed for dealing with multiple safety constraints in 2016; Swarm-based SafeOpt [8] was proposed in 2017 that applies a variant of particle swarm optimzation to SafeOpt; and, in 2018, StageOpt [17] was designed for optimizing the objective function in two independent stages of learning and optimization. The interested reader is referred to [11] for a detailed review of existing safe GP algorithms.

Here, we provide a high level description of selected safe GP algorithms – SafeOpt, modified SafeOpt, Safe-UCB [16] and modified Safe-UCB [5], which are the ones that will also be considered in the experimental study of this paper (alongside VA). These safe GP algorithms (also VA) are initialized with at least one known safe solution (i.e. a non-empty initial safe seed). However, while VA uses the information obtained from the initial safe seed to infer the safety of an offspring using the nearest neighbor method, SafeOpt and its variant Safe-UCB use the initial safe seed to establish a safe set (set of solutions inferred as being safe) based on the concept of *L*-Lipschitz continuity. This concept deems an input point

$\mathbf{x} \in D \subset \mathbb{R}^d$ as safe, if

$$l(\mathbf{x}^s) - L \cdot d(\mathbf{x}^s, \mathbf{x}) \geq h \tag{2}$$

for at least one input $\mathbf{x}^s \in S_{t-1}$, where S_{t-1} denotes a safe set estimated at the previous iteration step $t-1$ of the algorithm, $l(\mathbf{x}^s)$ is the lower bound of the predicted confidence interval for $\mathbf{x}^s$, $d(\cdot,\cdot)$ denotes distance between two input points (we used the Euclidean distance, which is often used in practice [6]), and L is the Lipschitz constant (see Section 1 in the Supplementary Material for technical details on the computation of the bounds). Here, S_0 consists of the initial safe seed(s). On the other hand, in modified SafeOpt and modified Safe-UCB, the initial safe seed is used for GP regression to find any input point $\mathbf{x} \in D \subset \mathbb{R}^d$ that meets

$$l(\mathbf{x}) \geq h\ . \tag{3}$$

When using safe GP algorithms, the search space is typically discretized uniformly (into e.g., 50×50 for two-dimensional input space [16]) to provide a finite set of input points for constructing safe set at each iteration step (i.e., S_t). Finally, an input point is selected in two different ways. In one case an input point whose width of predicted confidence interval is the maximum (modified SafeOpt) or upper confidence bound (UCB) is the greatest (modified Safe-UCB) is selected from the safe set for evaluation. On the other hand, the safe set is used for constructing further sets called maximizers set (set of input points likely to result in high output values when evaluated) and expanders set (set of input points that may increase the size of safe set when evaluated) where an input point is chosen from the union of the two sets for evaluation whose width of predicted confidence interval is the greatest (SafeOpt) or UCB is the maximum (Safe-UCB). Here, selecting an input point by the width of predicted confidence interval and by UCB criterion corresponds to adopting two predominant approaches in stochastic optimization, that is, *Bayesian experimental design* (exploring the objective function globally as efficiently as possible) and *multi-armed bandit paradigm* (maximizing cumulative reward), respectively [15].

In addition to differences in the working principles between safe EAs and safe GP algorithms (at least the ones described above), the two algorithm types vary in a few more further key aspects: Safe GP algorithms are designed to cope with noisy observations and require a discretized search space, while safe EA (VA) does not have a built-in noise-handling strategy nor require discretization. Also, SafeOpt and Safe-UCB assume that the Lischitz constant L is known, which is a strong assumptions to make in practice. Other safe GP algorithms vary in aspects, such as the method used to infer the safety level of a solution, generate solutions to evaluate next, and reliance on an initial safe seed and the Lipschitz continuity assumption [11].

4 EXPERIMENTAL SETUP

This section motivates the experimental setup used for benchmarking safe optimization algorithms. In particular, we motivate the choice of algorithms and their parameter settings, and explain how we simulate a safeOP as used in the subsequent experimental study.

Table 1: Algorithm parameter settings as used in the experimental study.

Algorithm	Parameter	Setting
Safe GP algorithms	β_t	2
	Lipschitz constant L	Maximum gradient on discretized domain
Safe EA and UnsafeEA	Population size μ	Number of initial safe seeds
	Offspring population size λ	Number of initial safe seeds
	Parental selection	Binary tournament
	Crossover operator	Uniform crossover
	Crossover probability p_c	0.8
	Mutation operator	Gaussian mutation, $\mu_m = 0$, $\sigma_m = 0.1$
	Mutation probability p_m	$1/d$
	Environmental selection	(μ+λ)-ES
All	Function evaluations	100

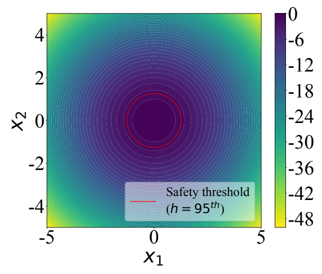
4.1 Algorithm selection and settings

Algorithm selection. The benchmark study will consider a safe EA representative, VA [10] augmented onto a generational EA with a $(\mu + \lambda)$-ES environmental selection strategy [3], and four safe GP algorithms: SafeOpt [16], Safe-UCB [16], modified SafeOpt [5], modified Safe-UCB [5]. As a baseline, we will also consider a generational EA with a $(\mu + \lambda)$-ES environmental selection strategy without the VA (we will refer to this algorithm as UnsafeEA), meaning this algorithm is blind to safety constraints. The set of algorithms selected allows us to compare the performance of two different working principles (EAs vs GPs). The selected algorithms are also well-known and widely used in their own communities (especially UnsafeEA and SafeOpt). Finally, with respect to the safe GPs, we have a good mix of algorithms varying in terms of their assumptions and input point selection criteria.

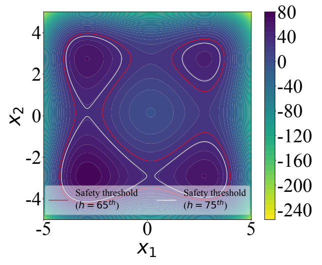
Algorithm parameter settings. Table 1 summarizes the algorithm parameters and their values as used in the experimental study. Most of these settings are standard settings as suggested in the original papers of the respective algorithms.

For VA and UnsafeEA, we set $\mu = \lambda =$ number of initial safe seed(s) provided. Both algorithms generate two offspring at a time by first using binary tournament selection to select two solutions from the current population, which then undergo uniform crossover ($p_c = 0.8$) and Gaussian mutation ($p_m = 1/d$, mean $\mu_m = 0$, standard deviation $\sigma_m = 0.1$); this process is repeated until λ offspring are generated. The variation operator settings were selected based on preliminary experimentation to achieve robust results.

With respect to safe GPs, to make SafeOpt and Safe-UCB more practical, we have modified the method for computing the upper



(a) Sphere function



(b) Styblinski-Tang function

Figure 1: Contour plots showing the objective function values (noiseless) for the (a) Sphere function and the (b) Styblinski-Tang function for $D \subset \mathbb{R}^2$. For both functions we also show the safety threshold h (as considered in the experimental study) bounding the safe regions in the search space. Note in (b) that the value of h controls connectiveness between safe search space regions centred around local optima.

and lower confidence interval, and also updated the way the Lipschitz constant, L, is estimated (see Section 1 in the Supplementary Material for the technical details).

As mentioned previously, while the safe GP algorithms have an in-built method for handling noise, the safe EA method (VA) and basic $(\mu + \lambda)$-ES do not. To be able to cope with noise meaningfully, we extend the two algorithms with a simple average scheme: any solution $\mathbf{x}$ that has been evaluated multiple times during search will use the average across the multiple noisy evaluations as the objective function value. To make this approach work we will need to keep track of the search history (previously evaluated solutions and their objective function values) of an algorithm. We also make sure that any duplicates of a solution in the population have the same objective function value.

For analysis of the results we keep track of the best-so-far objective function (BSF) value and total number of unsafe solutions evaluated at each evaluation step by an algorithm. Also, while the algorithms will be receiving and dealing with a noisy objective function value ($y(\mathbf{x})$), for the visualisations we will be using the corresponding true objective function value $f(\mathbf{x})$. If not stated otherwise, the results shown have been obtained across 20 independent algorithmic runs.

4.2 Test problems

As the focus of this paper is on building up a fundamental understanding of how EAs and other algorithms behave on SafeOPs, we do a deep dive analysis of two test problems varying in modality, on which we augment the notion of safety: the unimodal Sphere function and the multimodal Styblinski-Tang function. These two functions will allow us to understand the impact of different problem settings on the search behaviour as well as the global optimization behavior of the algorithms (in the case of the Styblinski-Tang function). The two test problems are shown visually in Figure 1 (for $d = 2$) and defined formally in the following.

The *Sphere function* [9] is defined as

$$f(\mathbf{x}) = f(\mathbf{x}^*) - ||\mathbf{x}_e||^2 \, , \tag{4}$$

where $\mathbf{x}^*$ is the location of the global maximum, $\mathbf{x}_e = \mathbf{x} - \mathbf{x}^*$, and $|| \cdot ||$ denotes the Euclidean norm. We set $\mathbf{x}^* = (0, 0)$ and $f(\mathbf{x}^*) = 0$.

The *Styblinski-Tang function* [18] is defined as

$$f(\mathbf{x}) = -\left(\frac{1}{2} \sum_{k=1}^{d} x_k^4 - 16x_k^2 + 5x_k \right) \, , \tag{5}$$

where d is the number of decision variables, the location of the global maximum is $\mathbf{x}^* = (-2.903534, \cdots, -2.903534)$, and $f(\mathbf{x}^*) = 39.16599 \times d$. This multimodal problem has three local maxima (one in bottom right, one in the top left, and one in the top right corner), and one global maximum (bottom left corner).

Note that both functions have no feasibility constraints, and for ease of visualisation of the results we set $d = 2$ for both problems.

4.3 Converting a problem into a SafeOP

To convert an optimization problem without safety constraints into a SafeOP, one needs to decide on mechanisms to (i) discretize the search space (as needed for safe GPs), (ii) set the safety threshold h, and (iii) generate the initial safe seed. We explain the mechanisms adopted here in turn.

Discretizing the search space. We set the search space for both test problems to $D = [-5, 5] \times [-5, 5]$. The search space is uniformly discretized into 500×500 using inspiration from [16], where a search space of $D = [0, 1] \times [0, 1]$ was discretized into 50×50 for SafeOpt and Safe-UCB.

Setting the safety threshold. Recall that a solution with an objective function below the safety threshold h is deemed as unsafe. The greater the value of h, the larger the portion of the search space deemed to be unsafe and thus the more constrained the search becomes. In this work, we set h such that we can investigate different more or less challenging search scenarios. We set h to be the percentile (i.e. between 0 and 100) of solutions (i.e., percentile value of the output values of input points on the discretized domain, in the remainder of this paper, percentile value denotes this concept) in the search space that are unsafe. That is, the lower the percentile, the fewer solutions are unsafe. For the sphere function, the safety threshold does not change the modality of the problem (it remains unimodal regardless of h) but simply controls the available safe search space because of the unimodal nature of the problem. This is different for the multimodal Styblinski-Tang function, which has three local and one global optima. Here we investigate two settings of h: a lower threshold value, which causes one local optimum to be isolated (i.e. the optimizer needs to traverse through an unsafe region in the search space to reach that local optimum or escape from it) from the other local optima, and a higher threshold value, resulting in all local optima being isolated from each other. The effect of the different safety thresholds (percentiles) on the two test problems is also shown in Figure 1b, and the safety thresholds considered in the study are shown in Table 2.

Generating the initial safe seed. In practice, the initial safe seeds would be known safe solutions to the problem at hand. To create an artificial SafeOP, we sample the initial safe seeds from input points in the discretized domain that meet $f(\mathbf{x}) - \sigma \cdot \beta \geq$

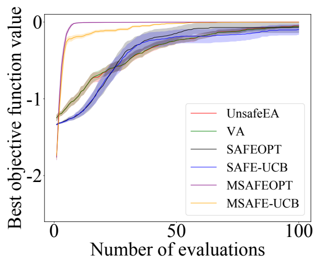
Table 2: Test problem settings as used in the experimental study.

Test problem	Problem parameter	Parameter setting
Sphere function	Safety threshold h	95th
Styblinski-Tang function	Safety threshold h	{65th, 75th}
Both functions	Nr of initial safe seeds	{2, 10}
	Search space dimension d	2
	Noise level σ	0.1

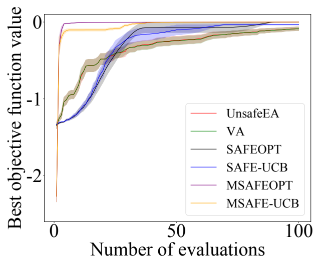
h, where σ and β denote the noise parameter in Eq. (1) and the parameter for the confidence interval of the noise, respectively. We set $\beta = 1.96$ to guarantee providing initial safe seeds that are safe from the noise (i.e., $y(\mathbf{x}) \geq h$, Eq. 1) to the algorithms for their initialization with 95% confidence. The standard deviation of the noise in Eq. (1) should be statistically estimated in practice, however, for simulation we have the freedom to set the level of noise; thus, we assume that $\sigma = 0.1$. Hence, if we know h, then we can use the condition that $f(\mathbf{x}) - \sigma \cdot \beta \geq h$ to sample an arbitrary number of initial safe seeds. We simulate the scenario where the optimizer is given 2 and 10 initial safe seeds, simulating a safeOP for which a decision maker has small and larger a priori knowledge regarding safety, respectively (see Table 2); consequently, our EA-based algorithms use also a population size of 2 and 10 (this is smaller than we would normally use in an EA but we need to bear in mind that we also allow only 100 objective function evaluations in total). While the initial safe seeds are generated at random for each of the 20 algorithmic runs, the same seeds are used across the benchmarked algorithms.

As shown in Figure 1b, there are three local and one global optima in the search space of Styblinski-Tang function, and we assume that there are scenarios on the sampling space of initial safe seeds: initializing from the top right local optimum which is isolated from the other local optima at any case (Scenario 1), from one of the local optima at the top left and bottom right corner which fall in connected safe search space region with global optimum when the level of safety threshold is low (Scenario 2), and from both of the top left and bottom right local optima (Scenario 3). The scenarios represent the difficulty of test problem; in scenario 1, knowledge on the search space is highly limited, in scenario 2, more knowledge on the search space is provided, and in scenario 3, the best environment for global optimization is given among the scenarios.

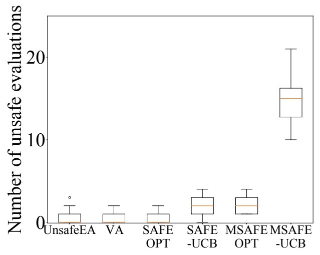
In scenario 3, one each and five each initial safe seeds are sampled from each local optimum, respectively (e.g., when the algorithms are initialized with two initial safe seeds, sample one initial safe seed from one local optimum, and the other initial safe seed from the other local optimum). We first investigate whether such scenarios on sampling space help or disturb global optimization, and then see the impact of safety threshold. Sampling initial safe seeds in a specific local optimum is performed by dividing the domain into four regions, region 1: $\mathbf{x}_1 \geq 0$ and $\mathbf{x}_2 \geq 0$, region 2: $\mathbf{x}_1 \leq 0$ and $\mathbf{x}_2 \geq 0$, region 3: $\mathbf{x}_1 \geq 0$ and $\mathbf{x}_2 \leq 0$, and region 4: $\mathbf{x}_1 \leq 0$ and $\mathbf{x}_2 \leq 0$, and limit the sampling space to the region(s) according to the given scenario.



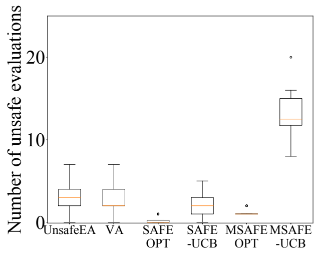
(a) BSF: Two initial safe seeds



(b) BSF: Ten initial safe seeds



(c) Unsafe: Two initial safe seeds



(d) Unsafe: Ten initial safe seeds

Figure 2: Plots show the mean best objective function value (BSF) as function of the number of function evaluations (top row) and distribution of the number of unsafe solutions evaluated across 20 algorithmic runs (bottom row) obtained for the various algorithms for the Sphere function (h = 95th). The plots on the left used 2 initial safe seeds, while 10 were used in the plots on the right-hand side.

5 RESULTS

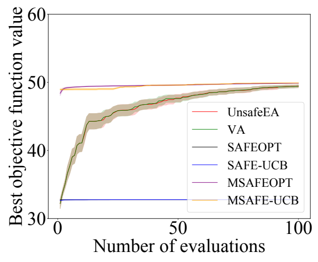
This section investigates the performance and search behaviour of a safeEA (VA), the four safeGPs, and an UnsafeEA for different problem parameters: locations of initial safe seeds, number of initial safe seeds, level of safety threshold, and safety budget. In the plots shown in this section and Supplementary Material, we will refer to modified SafeOpt [5] and modified Safe-UCB [5] as MSafeOpt, and MSafe-UCB, respectively.

An open-source repository containing object-oriented Python code for generating safeOP with different features (including the ones investigated here) and methods for the visualization can be downloaded from https://github.com/XXX.git.

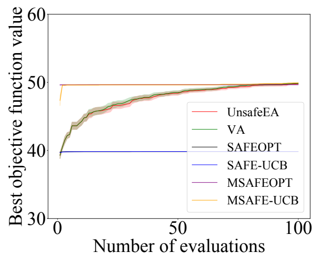
5.1 Impact of the number of initial safe seeds

Since the Sphere function is a unimodal function, we do not need to account for any search biases induced by local optima or other landscape features.

Figure 2 shows the BSF and number of unsafe solutions evaluated for the different algorithms for two initial safe seed settings (2 vs 10). In general, we observe that increasing the number of initial safe seeds is beneficial in terms of converging to an optimum (top row plots) more quickly and more robustly (as indicated by the



(a) BSF: h = 65th



(b) BSF: h = 75th

Figure 3: Plots show the mean best objective function value (BSF) as function of the number of function evaluations across 20 algorithmic runs obtained for the various algorithms for the Styblinski-Tang function using ten initial safe seeds. The algorithms were initialized with scenario 1. The plot on the left used h = 65th (i.e. 3 out of 4 local optima fall in connected safe search space region), while h = 75th (i.e. all local optima are in disconnected safe search space regions) was used in the right plot.

slightly lower standard error for 10 initial safe seeds). However, we can see that more initial safe seeds (bottom row of plots) results in more unsafe solutions being evaluated by the two EA-based methods (VA and UnsafeEA), while it has the opposite effect on the safeGPs except for Safe-UCB. The pattern of the EA-based methods is explained by the fact that for an initial safe seed of 2, we also have a population size of 2, and therefore there is a limited exploration power by the two algorithms reducing the chance of evaluating an unsafe solution. On the contrary, having more initial safe seeds allows the safeGPs to build a more accurate model of the search landscape and hence be more certain about the safety status of a solution, leading to fewer unsafe solutions being evaluated.

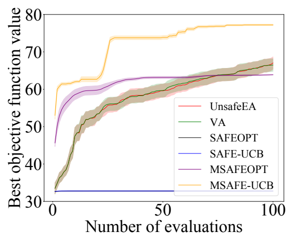
The similar performance of VA and UnsafeEA for the two initial safe seed settings may imply that in an environment where unsafe solutions are generally less likely to be evaluated, it is difficult to estimate the safety level of a solution by looking at the safety status of the nearest neighbor (as done by VA). However, VA shows, in general, better performance in terms of the number of unsafe evaluations (at least, equal but in most cases slightly better, see additional figures in Supplementary Material).

It is expected that the two algorithms, modified SafeOpt and modified Safe-UCB, outperform the original counterparts in terms of BSF at the cost of evaluating more unsafe solutions. This is because the modified versions are set up to be more explorative/risky, as they do not rely on L, see Eq. 3.

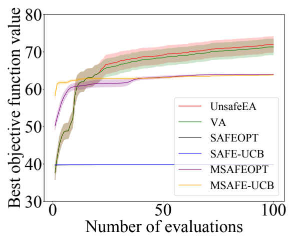
5.2 Impact of the Level of Safety Threshold

Let us now have a closer look at the impact of the safety threshold h on performance and search behaviour.

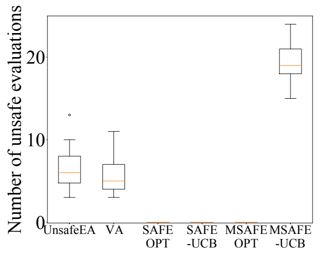
Considering the Styblinski-Tang function, when the algorithms are initialized with scenario 1, none of the algorithms performed global optimization because they were not able to jump out of that local safe region regardless of the level of safety threshold (see Figure 3). However, an extreme behavior was observed for scenario 2 when h = 65th percentile value (see Supplementary Material).



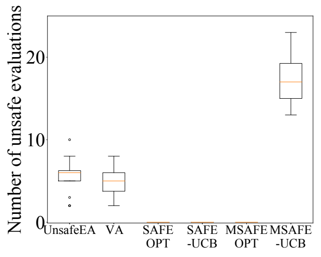
(a) BSF: h = 65th percentile



(b) BSF: h = 75th percentile



(c) Unsafe: h = 65th percentile



(d) Unsafe: h = 75th percentile

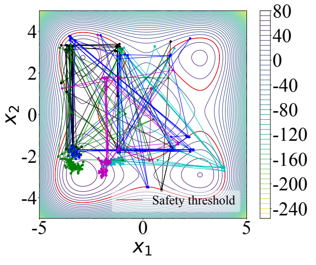
Figure 4: Plots show the mean best objective function value (BSF) as function of the number of function evaluations (top row) and distribution of the number of unsafe solutions evaluated across 20 algorithmic runs (bottom row) obtained for the various algorithms for the Styblinski-Tang function using ten initial safe seeds. Here, the initial safe seeds are sampled with scenario 3. The plots on the left used h = 65th percentile (i.e. the global and 2 out of 3 local optima fall in connected safe search space region), while h = 75th percentile (i.e. all local optima are in disconnected safe search space regions) was used in the right plots.

Modified Safe-UCB [5] performed global optimization (it discovered solutions of quality better than the local optima around which the initial safe seeds were placed) achieving great BSF value, but produced a lot of unsafe evaluations. However, it did not show any potential for global optimization when the level of safety threshold increased. On the other hand, the other algorithms did not perform global optimization in scenario 2 regardless of the level of safety threshold. A similar behavior was observed for scenario 3 when ten initial safe seeds are provided, that is, extreme behavior of modified Safe-UCB was observed when h = 65th percentile value, but it did not perform global optimization when h = 75th percentile value. However, in this scenario, we observe the potential of global optimization of another algorithm.

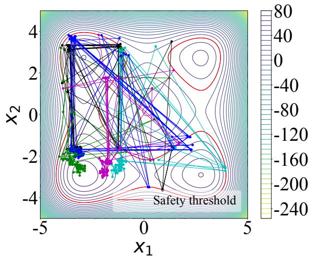
Figure 4 shows BSF and number of unsafe solutions evaluated for the Styblinski-Tang function using two different safety thresholds where the algorithms are initialized with ten initial safe seeds. We can observe that increasing the safety threshold level h (i.e. increasing the presence of unsafe solutions) improves the BSF discovered by VA and UnsafeEA, while reducing the likelihood for evaluating unsafe solutions. This may happen as VA is initialized with initial safe seeds whose output values are significantly greater than the initial safe seeds sampled when h = 65th percentile value. Also, this implies that connectivity of the local optima is not a necessary condition for global optimization of VA, and having diverse information about the safe regions around the global optimum is likely to help global optimization. For example, all local optima are isolated with each other when $h = 75^{\text{th}}$ percentile value, but VA is able to perform global optimization when it is initialized at both local optima which were connected to the global optimum when $h = 65^{\text{th}}$ percentile value. On the other hand, connectivity may be interpreted as the key factor for global optimization of modified Safe-UCB because it performed global optimization when $h = 65^{\text{th}}$ percentile value while producing almost equal number of unsafe evaluations to those produced when $h = 75^{\text{th}}$ percentile value, which is a test problem where the algorithm was unable to perfrom global optimization. An interesting finding is that while SafeOpt [16] and Safe-UCB [16] performed well in terms of BSF and unsafe solutions evaluated for the Sphere function, the two algorithms are not able to improve the BSF significantly following initialization on Styblinski-Tang function. It may happen due to the limitation of estimating Lipschitz constant. We estimated the Lipschitz constant over the whole search space (see Section 1 in Supplementary Material), and as shown in Figure 1b, the gradient around the edge of the search space is much greater than the gradient in the safe regions. This results in highly risk-averse attitude of SafeOpt and Safe-UCB. We also observe that modified SafeOpt is not evaluating any unsafe solutions when compared to the Sphere function regardless of the value of h.

Figure 5 provides the search trajectory of the first five runs of each algorithm initialized with ten initial safe seeds sampled with scenario 3 where $h = 65^{\text{th}}$ percentile value. When we investigate the trajectories of EA-based optimizers, we can see that they use the information around the two local optima actively, e.g., the optimizers traverse the two optima actively and finally explore and exploit the global optimum, as shown in Figure 5a and Figure 5b. Trajectories of modified Safe-UCB, which is the other algorithm that performed global optimization, show that the algorithm explored the safe region very actively. However, when we see the other algorithm that performed well (at least not stuck at a low BSF value), modified SafeOpt quickly converge to one local optimum. Thus, the information given in initial safe seeds about safety in the two local optima is not fully used. What is worse is that SafeOpt and Safe-UCB do traverse the two local optima, however, they just stuck at some input points whose quality of solution is significantly low.

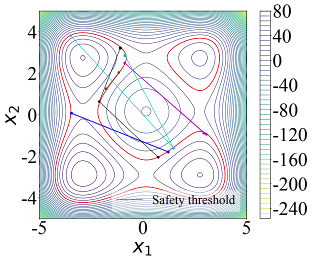
As expected, as presented in Figure 4, when comparing the results of the EA and GP-based optimizers, we note that the EA-based optimizers (both VA and UnsafeEA) are doing significantly better than the GP-based optimizers in terms of BSF as the safety threshold increases (i.e. more of the search spaces becomes unsafe). We do not observe negative impact in terms of the number of unsafe evaluations along the increase of the level of safety threshold, and EA-based optimizers produce more unsafe evaluations than GP-based optimizers overall (except for modified Safe-UCB that showed extreme behavior). This is due to the combination of having few initial safe seeds combined with the drive of EA to explore more globally.



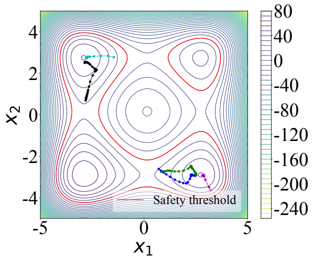
(a) UnsafeEA



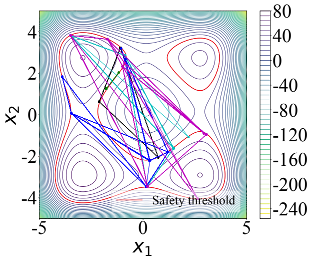
(b) VA



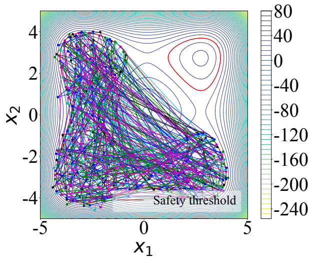
(c) SafeOpt



(d) Modified SafeOpt



(e) Safe-UCB

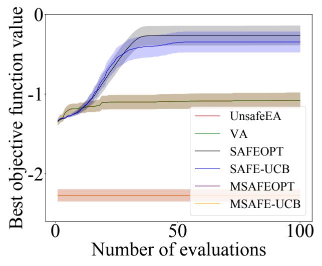


(f) Modified Safe-UCB

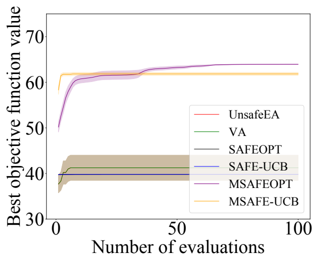
Figure 5: Plots show trajectories of the algorithms when initialized with ten initial safe seeds sampled with scenario 3 ($h = 65^{\text{th}}$). (a) Trajectory of 5 runs of UnsafeEA. (b) Trajectory of 5 runs of VA. (c) Trajectory of 5 runs of SafeOpt. (d) Trajectory of 5 runs of modified SafeOpt. (e) Trajectory of 5 runs of Safe-UCB. (f) Trajectory of 5 runs of modified Safe-UCB.

5.3 Budget on Unsafe Evaluations

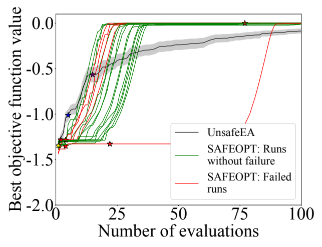
Here, we investigate the impact of budget on unsafe evaluations. For this, let us compare the results obtained by the algorithms for a setting without a limit on the safety budget (shown in Figures 2b and 4b), with a zero safety budget setting (Figure 6). We consider the Sphere function first. For a zero safety budget (Figure 6a), we observe that several algorithms (modified SafeOpt, modified Safe-UCB, and VA) cannot match their performance obtained in an environment without a limit on the safety budget (which was shown in Figure 2b). This is because a zero safety budget causes these algorithms to terminate an optimization run early, namely as soon as the first unsafe solution has been evaluated. SafeOpt [16] and Safe-UCB [16], which are provided with more information about the objective function (i.e., Lipschitz constant), are the only two algorithms considered that efficiently converge to the optimum on the Sphere function.



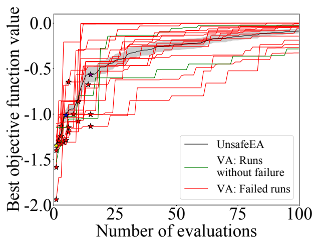
(a) Sphere function



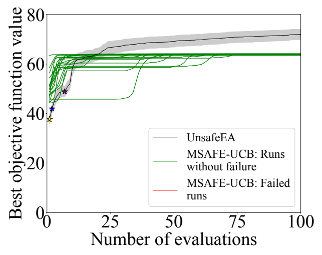
(b) Styblinski-Tang function



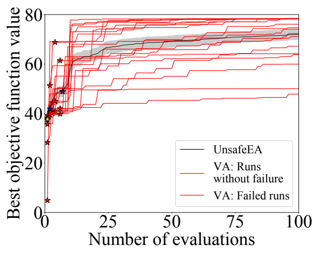
(c) Sphere function - SafeOpt



(d) Sphere function - VA



(e) Styblinski-Tang function - modified SafeOpt



(f) Styblinski-Tang function - VA

Figure 6: Plots show the mean best objective function value (BSF) across 20 algorithmic runs as function of the number of function evaluations (top row) and BSF of each algorithmic run of SafeOpt and VA (middle row), on sphere function, and modified SafeOpt and VA (bottom row), on Styblinski-Tang function, as function of the number of function evaluations. Here, 10 initial safe seeds are given to the algorithms, the budget on unsafe evaluation is zero, and $h = 95^{th}$ and $h = 75^{th}$ for sphere and Styblinski-Tang function, respectively. (c, d, e, f) Mean and mean ± standard error of UnsafeEA are presented as black solid line and an area around it, respectively. Red and green solid lines are runs showing at least one unsafe evaluation and without failure, respectively. The red stars show when the first failure happened for failed runs and yellow, blue, and purple stars inform the minimum, median, and maximum evaluation step of the first failure of all runs of UnsafeEA.

When zero budget on unsafe evaluations is applied to optimizing Styblinski-Tang function (Figure 6b), VA shows extremely poorer results, and modified SafeOpt and modified Safe-UCB (in comparison) become the best algorithms (see Figure 4b). It is because the first failure happens too early for VA as shown in Figure 6f. As mentioned in Section 5.2, VA shows a potential of global optimization, however, it costs to evaluate unsafe solutions, and the figure shows that the failure happens from the early stage of optimization. In contrast, as shown in Figure 5d, modified SafeOpt has a tendency to converge to a local optimum, rather than performing global optimization.

The results of each of the 20 algorithmic runs of the best performing algorithms on Sphere and Styblinski-Tang function are presented in Figure 6c (SafeOpt) and Figure 6e (modified SafeOpt) respectively. In comparison, we also show the same data for VA on the Sphere function (Figure 6d) and Styblinski-Tang function (Figure 6f). We can observe that GP-based optimizers in Figure 6c and 6e are relatively more invariant to the location of initial safe seeds, however, VA shows a significant level of variation in the results across the 20 runs.

6 CONCLUSION AND FUTURE RESEARCH

The violation of safety constraints in an optimization problem causes an irrevocable loss to the system, e.g. breakage of an expensive hardware kit or loss of a human life. This paper has investigated such safe optimization problems (safeOPs) by analyzing the performance of two different types of optimization algorithms: EA-based vs GP-based algorithms. This was the largest benchmark study in the safe optimization literature. We also considered a baseline EA, which was blind to safety constraints. Furthermore, we have developed a feature-rich open-source Python framework for (i) converting any optimization problem into a safeOP containing different user-specified properties, (ii) benchmarking different optmzation algorithms, and (iii) visualizing the performance in different ways.

The focus of our experimental study was to determine whether EA-based methods can compete with modern GP-based methods and understand the search behaviour of all algorithms considered for problems of varying complexity.

Overall, we observed that the performance of a safe optimization algorithms is affected by the modality of the fitness landscape, as well as key problem parameters, such as parameters governing the location of initial safe seeds, the number of initial safe seeds, the level of safety threshold, and the budget on unsafe evaluations. Finally, we observed that an EA blind to safety constraints did not perform this badly; while we cannot say that EAs are safe optimizers generally, we can certainly say that EAs have potential to become safe optimizers, e.g. when augmented with a pre-screening methods as employed by VA.

We suggest several topics for future research based on the above observations. An obvious topic is to focus on developing more efficient safe optimization algorithms, and then testing them for different problem settings, similar to how we have done here. This will help build up a fundamental understanding about this so important problem type. There is also a need to expand the open-source Python framework that we have made available here to reduce the barrier for new scientists to enter this field and speed up development. Finally, it would be important to understand better the landscape properties of real safe optimization problems and how they vary from the artificial ones created here. This can provide indication about the suitability of our methods in practice.

REFERENCES

[1] Richard Allmendinger and Joshua D. Knowles. 2011. Evolutionary Search in Lethal Environments. In *International Conference on Evolutionary Computation Theory and Applications*. SciTePress, 63–72. https://doi.org/10.5220/0003673000630072 arXiv:https://www.scitepress.org/papers/2011/36730/36730.pdf

[2] François Bachoc, Céline Helbert, and Victor Picheny. 2020. Gaussian process optimization with failures: Classification and convergence proof. *Journal of Global Optimization* (2020). https://doi.org/10.1007/s10898-020-00920-0

[3] Thomas Baeck, DB Fogel, and Z Michalewicz. 2000. *Evolutionary Computation 1: Basic Algorithms and Operators*. Vol. 1. CRC Press.

[4] Felix Berkenkamp, Andreas Krause, and Angela P. Schoellig. 2016. Bayesian Optimization with Safety Constraints: Safe and Automatic Parameter Tuning in Robotics. *Arxiv preprint arXiv:1602.04450* (2016). http://arxiv.org/abs/1602.04450

[5] Felix Berkenkamp, Angela P. Schoellig, and Andreas Krause. 2016. Safe controller optimization for quadrotors with Gaussian processes. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 491–496. https://doi.org/10.1109/ICRA.2016.7487170

[6] Erdem Bıyık, Jonathan Margoliash, Shahrouz Ryan Alimo, and Dorsa Sadigh. 2019. Efficient and Safe Exploration in Deterministic Markov Decision Processes with Unknown Transition Models. In *2019 American Control Conference (ACC)*. IEEE, 1792–1799. https://doi.org/10.23919/ACC.2019.8815276

[7] Carlos A Coello Coello. 2002. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer methods in applied mechanics and engineering* 191, 11-12 (2002), 1245–1287.

[8] Rikky R. P. R. Duivenvoorden, Felix Berkenkamp, Nicolas Carion, Andreas Krause, and Angela P. Schoellig. 2017. Constrained Bayesian Optimization with Particle Swarms for Safe Adaptive Controller Tuning. *IFAC-PapersOnLine* 50, 1 (2017), 11800–11807. https://doi.org/10.1016/j.ifacol.2017.08.1991

[9] Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. 2009. *Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions*. Research Report RR-6829. INRIA. https://hal.inria.fr/inria-00362633

[10] H. Kaji, Kokolo Ikeda, and Hajime Kita. 2009. Avoidance of constraint violation for experiment-based evolutionary multi-objective optimization. In *Proceedings of the 2009 Congress on Evolutionary Computation (CEC 2009)*. IEEE Press, Piscataway, NJ, 2756–2763. https://doi.org/10.1109/CEC.2009.4983288

[11] Youngmin Kim, Richard Allmendinger, and Manuel López-Ibáñez. 2021. Safe Learning and Optimization Techniques: Towards a Survey of the State of the Art. In *Trustworthy AI - Integrating Learning, Optimization and Reasoning*, Fredrik Heintz, Michela Milano, and Barry O'Sullivan (Eds.). Springer International Publishing, Cham, 123–139.

[12] Zbigniew Michalewicz. 1995. A Survey of Constraint Handling Techniques in Evolutionary Computation Methods. In *Evolutionary Programming*.

[13] Matthieu Sacher, Régis Duvigneau, Olivier Le Maitre, Mathieu Durand, Elisa Berrini, Frédéric Hauville, and Jacques-André Astolfi. 2018. A classification approach to efficient global optimization in presence of non-computable domains. *Structural and Multidisciplinary Optimization* 58, 4 (2018), 1537–1557. https://doi.org/10.1007/s00158-018-1981-8

[14] Mark Schillinger, Benjamin Hartmann, Patric Skalecki, Mona Meister, Duy Nguyen-Tuong, and Oliver Nelles. 2017. Safe active learning and safe Bayesian optimization for tuning a PI-controller. *IFAC-PapersOnLine* 50, 1 (2017), 5967–5972. https://doi.org/10.1016/j.ifacol.2017.08.1258

[15] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. 2010. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. In *Proc. International Conference on Machine Learning (ICML)*.

[16] Yanan Sui, Alkis Gotovos, Joel W. Burdick, and Andreas Krause. 2015. Safe Exploration for Optimization with Gaussian Processes. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015*, Francis Bach and David Blei (Eds.), Vol. 37. 997–1005. arXiv:http://proceedings.mlr.press/v37/sui15.html

[17] Yanan Sui, Vincent Zhuang, Joel W. Burdick, and Yisong Yue. 2018. Stagewise Safe Bayesian Optimization with Gaussian Processes. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018 (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer G. Dy and Andreas Krause (Eds.). PMLR, 4788–4796. arXiv:http://proceedings.mlr.press/v80/sui18a.html

[18] D. Zubanovic, A. Hidic, A. Hajdarevic, N. Nosovic, and S. Konjicija. 2014. Performance analysis of parallel master-slave Evolutionary strategies (μ,λ) model python implementation for CPU and GPGPU. In *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 1609–1613. https://doi.org/10.1109/MIPRO.2014.6859822

Are Evolutionary Algorithms Safe Optimizers?

Supplementary Material

YOUNGMIN KIM, Alliance Manchester Business School, University of Manchester, UK
RICHARD ALLMENDINGER, Alliance Manchester Business School, University of Manchester, UK
MANUEL LÓPEZ-IBÁÑEZ*, Alliance Manchester Business School, University of Manchester, UK

Contents

List of Figures

*Also with School of Computer Science, University of Málaga.

1 PARAMETER SETTING FOR SAFE GP ALGORITHMS

SafeOpt [9] and modified SafeOpt [2] select an input point whose width of predicted confidence interval is the maximum for evaluation (*Bayesian experimental design* [8]), whereas UCB criterion is applied in Safe-UCB [9] and modified Safe-UCB [2] (*multi-armed banit paradigm* [8]). Here, predicted confidence interval can be represented by lower (l_t) and upper (u_t) bound:

$$l_t = \mu_{t-1}(\mathbf{x}) - \beta_t \sigma_{t-1}(\mathbf{x}),\ u_t = \mu_{t-1}(\mathbf{x}) + \beta_t \sigma_{t-1}(\mathbf{x}) \tag{1}$$

where β_t is a parameter that decides the width of the predicted confidence interval and σ_{t-1} is the predicted standard deviation at point $\mathbf{x}$ [7], at t^{th} iteration step [6]. There are two different approaches in selecting β_t in Eq. (1). In SafeOpt, β_t is set as an iteration-varying scalar, while modified SafeOpt used a fixed constant over iterations ($\beta_t = 2$). As seen in [1], setting β_t as a fixed constant over iterations approximately means that we apply a certain failure probability per iteration, not over all iterations. Thus, it would be ideal if the iteration-varying scalar can be estimated at each iteration step for β_t, however, there is a strict assumption for applying it to estimating predicted confidence interval. SafeOpt [9] adopted a study [8], which decides the tightness of predicted confidence interval by controlling the value of β_t at each iteration step. In the study [8], the parameter β_t at t^{th} iteration step is estimated as,

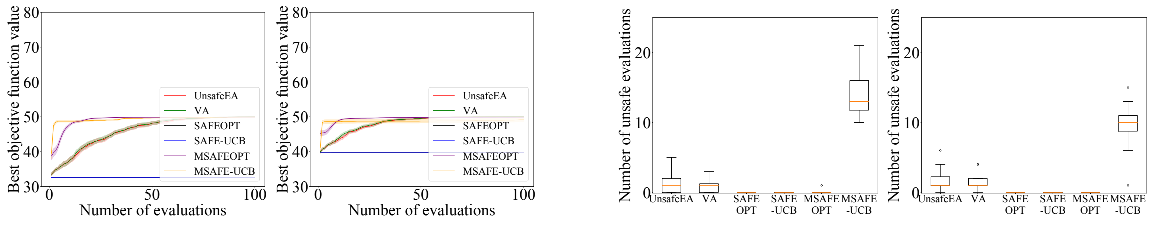
$$\beta_t = 2B + 300\gamma_t \log^3(t/\delta) \tag{2}$$

where B is a bound on the *reproducing kernel Hilbert space* (RKHS) norm, γ_t is the maximal mutual information which can be obtained from t samples about the GP prior, and δ is a probability of failure allowed [8, 9]. B in Eq. (2) is available under the assumption that an objective function in an optimization problem lies in RKHS. Estimating β_t based on Eq. (2) was appropriate in [9], as the experiments on synthetic data in that paper were made on random objective functions sampled from a zero-mean GP with squared-exponential kernel. However, it is difficult to assume that an objective function in a test problem, created while assuming the real-world problem, lies in RKHS, and thus, we could not apply Eq. (2) for estimating β_t. Instead, we set β_t equal to two for the experiments, which was used for modified SafeOpt and is one of the typical choices used for estimating predicted confidence interval [4]. For details about RKHS, please see [5], and to see details about Eq. (2) (e.g., how the equation was deducted, how to estimate maximal mutual information, etc.), please see [3, 8, 9].

In [9], the gradient of several random functions, that were sampled from the same GP used for sampling objective functions for experiments on synthetic data in that study, was used for estimating Lipschitz constant. However, we cannot sample any function to estimate Lipschitz constant for our experiments; thus, we compute gradient directly from our objective functions on the discretized domain, estimate Lipschitz constant (by the definition of L-Lipschitz continuity, we set the maximum absolute value of gradient observed over the discretized domain as a Lipschitz constant), and provide it to SafeOpt and Safe-UCB. Hence, the results of our benchmarking experiments are made with perfect information given for estimating the gradient of our objective function; thus, please bear in mind that in the condition where less information on gradient of an objective function is given, the result can be worse than ours, e.g., more safety constraint violations and slower optimization process (or even stuck at certain area), when smaller and greater value than our Lipschitz constant value is given as Lipschitz constant, respectively.

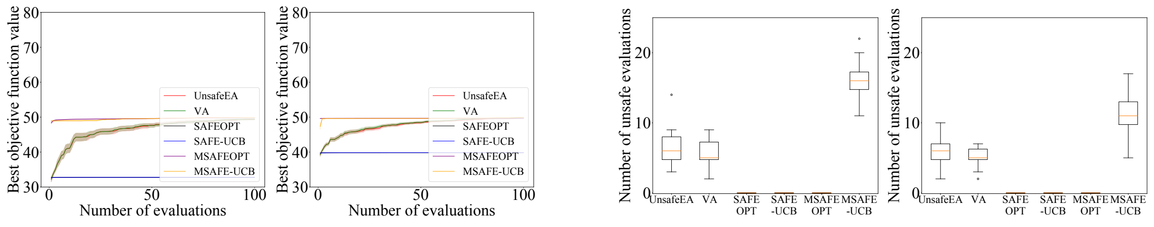
2 SUPPLEMENTARY FIGURES: STYBLINSKI-TANG FUNCTION

Here, plots for each scenario of sampling space of initial safe seeds for Styblinski-Tang function are provided.



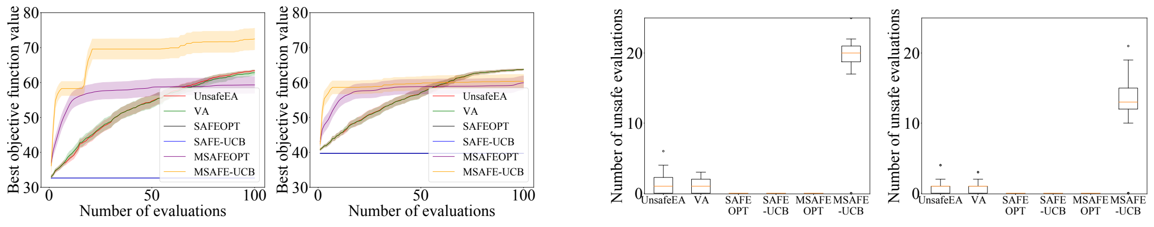
(a) BSF: h = 65th (b) BSF: h = 75th (c) Unsafe: h = 65th (d) Unsafe: h = 75th

Fig. 1. Plots show the average best objective function value (BSF) as function of the number of function evaluations and distribution of the number of unsafe solutions evaluated across 20 algorithmic runs on the Stylinski-Tang function (2 initial safe seeds). Initial safe seeds were sampled with scenario 1.



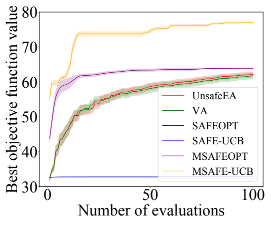
(a) BSF: h = 65th (b) BSF: h = 75th (c) Unsafe: h = 65th (d) Unsafe: h = 75th

Fig. 2. Plots show the average best objective function value (BSF) as function of the number of function evaluations and distribution of the number of unsafe solutions evaluated across 20 algorithmic runs on the Stylinski-Tang function (10 initial safe seeds). Initial safe seeds were sampled with scenario 1.

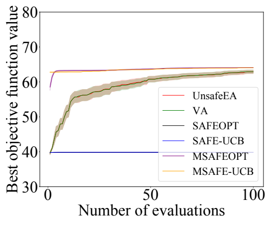


(a) BSF: h = 65th (b) BSF: h = 75th (c) Unsafe: h = 65th (d) Unsafe: h = 75th

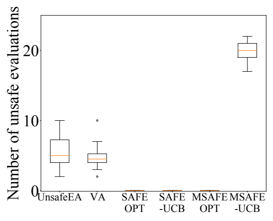
Fig. 3. Plots show the average best objective function value (BSF) as function of the number of function evaluations and distribution of the number of unsafe solutions evaluated across 20 algorithmic runs on the Stylinski-Tang function (2 initial safe seeds). Initial safe seeds were sampled with scenario 2.



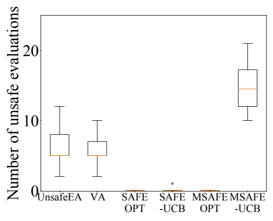
(a) BSF: $h = 65^{\text{th}}$



(b) BSF: $h = 75^{\text{th}}$

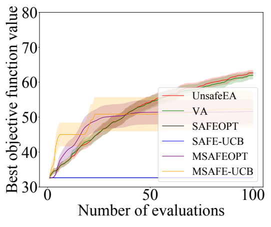


(c) Unsafe: $h = 65^{\text{th}}$

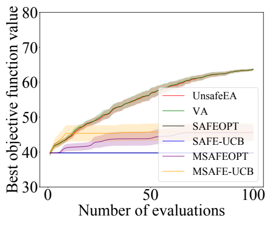


(d) Unsafe: $h = 75^{\text{th}}$

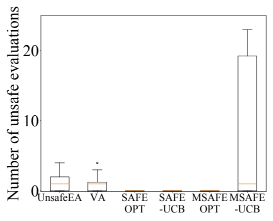
Fig. 4. Plots show the average best objective function value (BSF) as function of the number of function evaluations and distribution of the number of unsafe solutions evaluated across 20 algorithmic runs on the Stylinski-Tang function (10 initial safe seeds). Initial safe seeds were sampled with scenario 2.



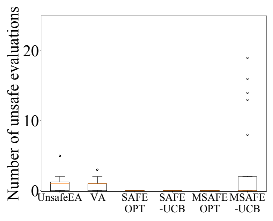
(a) BSF: $h = 65^{\text{th}}$



(b) BSF: $h = 75^{\text{th}}$

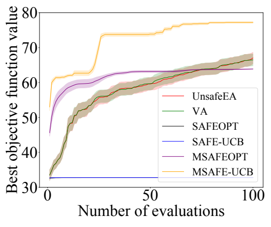


(c) Unsafe: $h = 65^{\text{th}}$

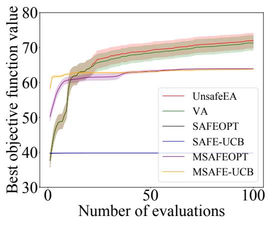


(d) Unsafe: $h = 75^{\text{th}}$

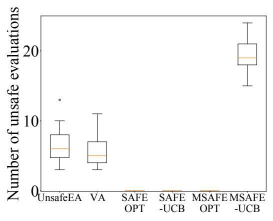
Fig. 5. Plots show the average best objective function value (BSF) as function of the number of function evaluations and distribution of the number of unsafe solutions evaluated across 20 algorithmic runs on the Stylinski-Tang function (2 initial safe seeds). Initial safe seeds were sampled with scenario 3.



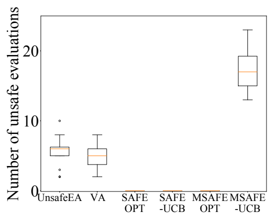
(a) BSF: $h = 65^{\text{th}}$



(b) BSF: $h = 75^{\text{th}}$



(c) Unsafe: $h = 65^{\text{th}}$



(d) Unsafe: $h = 75^{\text{th}}$

Fig. 6. Plots show the average best objective function value (BSF) as function of the number of function evaluations and distribution of the number of unsafe solutions evaluated across 20 algorithmic runs on the Stylinski-Tang function (10 initial safe seeds). Initial safe seeds were sampled with scenario 3.

REFERENCES

[1] Felix Berkenkamp, Andreas Krause, and Angela P. Schoellig. 2016. Bayesian Optimization with Safety Constraints: Safe and Automatic Parameter Tuning in Robotics. *Arxiv preprint arXiv:1602.04450* (2016). http://arxiv.org/abs/1602.04450

[2] Felix Berkenkamp, Angela P. Schoellig, and Andreas Krause. 2016. Safe controller optimization for quadrotors with Gaussian processes. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 491–496. https://doi.org/10.1109/ICRA.2016.7487170

[3] Sayak Ray Chowdhury and Aditya Gopalan. 2017. On Kernelized Multi-Armed Bandits. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70* (Sydney, NSW, Australia) *(ICML'17)*. JMLR.org, 844–853.

[4] Rikky R. P. R. Duivenvoorden, Felix Berkenkamp, Nicolas Carion, Andreas Krause, and Angela P. Schoellig. 2017. Constrained Bayesian Optimization with Particle Swarms for Safe Adaptive Controller Tuning. *IFAC-PapersOnLine* 50, 1 (2017), 11800–11807. https://doi.org/10.1016/j.ifacol.2017.08.1991

[5] Arthur Gretton. 2013. Introduction to rkhs, and some simple kernel algorithms. *Adv. Top. Mach. Learn. Lecture Conducted from University College London* 16 (2013).

[6] Youngmin Kim, Richard Allmendinger, and Manuel López-Ibáñez. 2021. Safe Learning and Optimization Techniques: Towards a Survey of the State of the Art. In *Trustworthy AI - Integrating Learning, Optimization and Reasoning*, Fredrik Heintz, Michela Milano, and Barry O'Sullivan (Eds.). Springer International Publishing, Cham, 123–139.

[7] Eric Schulz, Maarten Speekenbrink, and Andreas Krause. 2018. A tutorial on Gaussian process regression: Modelling, exploring, and exploiting functions. *Journal of Mathematical Psychology* 85 (Aug. 2018), 1–16. https://doi.org/10.1016/j.jmp.2018.03.001

[8] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. 2010. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. In *Proc. International Conference on Machine Learning (ICML)*.

[9] Yanan Sui, Alkis Gotovos, Joel W. Burdick, and Andreas Krause. 2015. Safe Exploration for Optimization with Gaussian Processes. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015*, Francis Bach and David Blei (Eds.), Vol. 37. 997–1005. arXiv:http://proceedings.mlr.press/v37/sui15.html