Combining Constraint Solving and Bayesian Techniques for System Optimization

Franz Brauße[1] , **Zurab Khasidashvili**[2] , **Konstantin Korovin**[1]
[1]The University of Manchester, UK
[2]Intel, Israel
{franz.brausse, konstantin.korovin}@manchester.ac.uk , zurab.khasidashvili@intel.com

Abstract

Application domains of Bayesian optimization include optimizing black-box functions or very complex functions. The functions we are interested in describe complex real-world systems applied in industrial settings. Even though they do have explicit representations, standard optimization techniques fail to provide validated solutions and correctness guarantees for them. In this paper we present a combination of Bayesian optimization and SMT-based constraint solving to achieve safe and stable solutions with optimality guarantees.

1 Introduction

Bayesian optimization (BO) [Mockus, 1975; Frazier, 2018] is a popular technique for optimizing an objective function f, which is done through searching for input points $\boldsymbol{x}$ such that $f(\boldsymbol{x})$ approximates the maximum $\max f$. These solutions are presumed to be near-optimal but BO solvers do not provide formal guarantees on the accuracy and it may well happen that the solution is arbitrary far from the real solution. BO does not require an explicit representation of f, and instead search for near-optimal points is based on sampling the values of f at a limited number of input points. BO is therefore used mainly for optimizing functions whose evaluation is expensive. In particular, BO is often used for hyper-parameter optimization when training machine learning models [Snoek *et al.*, 2012]. During the last few decades, BO has been used extensively for designing engineering systems [Mockus, 1989].

BO iteratively builds a statistical model of $f(\boldsymbol{x})$ usually from a prior distribution defined by a Gaussian process [Rasmussen and Williams, 2006]. At each iteration i the current model is used to select the most promising candidate point $\boldsymbol{x}_i$ to evaluate $f(\boldsymbol{x}_i)$. This evaluation is used to update the posterior belief of the model. This process is repeated until some bound on the number of iterations is reached.

BO behaves well in practice and usually can find near-optimal points in just a few iterations. Nevertheless BO has its limitations. In particular, since BO is based on statistical approximations, there are no formal guarantees that the result achieved after a finite number of iterations is actually optimal or even close to optimal. Another limitation of BO is that even if the found point is near optimal, the solution may not be stable: there may be close points on which the value of the objective function is very different from the optimum. Such solutions are undesirable in many applications which require regions around solutions to be also near optimal. This motivates the combination of BO with SMT-based constraint solving to achieve safe and stable near-optimal solutions which we develop in this paper.

In this work we develop optimization techniques for systems modeled using real-valued functions. In many cases such functions are approximated using neural networks which are also covered in our framework. In this context, the assumption that the objective function is very expensive to sample is no longer valid. Instead, achieving tighter approximations to the maximum $\max f$ becomes more important even if computing near-optimal points that further improve the approximation accuracy becomes significantly more expensive. Besides, in this context, it is important to guarantee that the computed near-optimal point $\boldsymbol{x}$ satisfies the design constraints and it is robust [Beyer and Sendhoff, 2007] in the sense that small perturbations to $\boldsymbol{x}$ still yield legal near-optimal points for the objective function f.

We propose an optimization algorithm GEAROPT$_\delta$-BO which combines SMT-based constraint solving and BO and has the following properties:

- *Safety:* The computed near-optimal points are feasible (satisfy design constraints). In BO the feasibility of computed near-optimal solution is achieved by sampling the objective function and evaluating the constraints [Gardner *et al.*, 2014]. In contrast, in our approach we use the explicit representation of the constraints and of the objective function to guide the search for a feasible near-optimal point.
- *Accuracy:* Our algorithm can find safe near-optimal points for which the value of the objective function f is within a predefined distance from the real maximum, $\max f$. This is enabled by the fact that in our problem setting the objective function and the constraints are given explicitly, thus we can utilize SMT solving to precisely analyse them, whereas purely probabilistic methods cannot provide full guarantees due to a limited number of sampling of f.
- *Stability:* Furthermore, safe near-optimal points computed by our algorithm are *stable* in the sense that after

perturbation of inputs within user-specified regions:

1. the safety constraints remain valid; and
2. the output remains within the near-optimal range.

Stability is a critical requirement from analog devices because the inputs and the output can be perturbed due to uncertainties in the environment such as uncertain operating conditions, design parameter tolerances or actuator imprecisions [Beyer and Sendhoff, 2007]. This concept is studied in the BO setting as *robustness* [Bogunovic *et al.*, 2018; Sanders *et al.*, 2019; Cardelli *et al.*, 2019], where it can be estimated with high confidence but cannot be proven formally.

The paper is organized as follows. In the next section we introduce the notation. In Section 3 we recall the basics of BO and in Section 4 we define the stability and accuracy requirements for near-optimal solutions. In Section 5, we present our optimization algorithm GEAROPT$_\delta$-BO, which combines the strengths of the constraint solving algorithm GEARSAT$_\delta$ [Brauße *et al.*, 2020] and reasoning with probabilities (Bayesian inference). We evaluated GEAROPT$_\delta$-BO on industrial examples coming from optimization of microprocessor design at Intel where stability is an essential requirement. We conclude in Section 7.

2 Preliminaries

Given a function $f : A \to B$, by $\operatorname{dom} f$ we denote its domain A. Vectors $(x_1, \ldots, x_n)$ may occur in the abbreviated form $\boldsymbol{x}$. Given $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n$ with $a_i \leq b_i$ for all i, by $[[\boldsymbol{a}, \boldsymbol{b}]]$ we denote the Cartesian product $\times_i [a_i, b_i]$ of their componentwise closed intervals $[a_i, b_i]$.

In this paper we consider formulas over $\langle \mathbb{R}, 0, 1, \mathcal{F}, P \rangle$, where P are the usual order predicates $<, \leq, =$, etc. and $\mathcal{F}$ contains addition, multiplication with rational constants and can also contain non-linear functions supported by SMT solvers including polynomials, transcendental functions such as combinations of sine, cosine, exponentials, solutions of differential equations and more generally computable functions [de Moura and Bjørner, 2008; Brauße *et al.*, 2019; Brauße *et al.*, 2021; Cimatti *et al.*, 2013; Gao *et al.*, 2012]. We extend functions $\mathcal{F}$ by functions *definable* by formulas: $\mathcal{F}_D$, i.e., we assume $f \in \mathcal{F}_D$ is represented by a formula $F(x_1, \ldots, x_n, y)$ over variables $x_1, \ldots, x_n$ corresponding to the n inputs and y corresponding to the output $f(x_1, \ldots, x_n)$. We assume that satisfiability of such formulas is decidable or more generally δ-decidable [Gao *et al.*, 2012]. Let us note that even when basic functions $\mathcal{F}$ contain just linear functions, $\mathcal{F}_D$ will contain, e.g., functions represented by neural networks with ReLU activation functions. Optimization of functions represented by neural networks is one of the main motivations behind this work.

Throughout, x, y denote variables in formulas while a, b, c, d, e, T stand for rational constants; both forms may be indexed. Whenever we use a norm $\|\cdot\|$, we refer to the Chebyshev norm $(x_1, \ldots, x_n) \mapsto \max\{|x_1|, \ldots, |x_n|\}$.

3 Bayesian Optimization

A basic form of BO algorithm involves two primary components: a method for statistical inference, typically Gaussian process regression [Rasmussen and Williams, 2006]; and an acquisition function for deciding where to sample next. The most popular choices for an acquisition function include expected improvement towards the optimum, knowledge gradient, and entropy search. BO iteratively builds a statistical model of an objective function $f(\boldsymbol{x})$ from a prior distribution defined by a Gaussian process. At each iteration i the current model is used to select the most promising candidate point $\boldsymbol{x}_i$ to evaluate the objective function $f(\boldsymbol{x}_i)$. This evaluation is used to update the posterior belief of the model. This process is repeated until bound $MaxIter$ on the number of iterations is reached.

The basic optimization algorithm for maximizing a black-box function $f : \mathbb{R}^n \to \mathbb{R}$, depicted in Algorithm 1, which shows a BO solver $A^{\max}$ with the following interface.

INIT Inputs: $\boldsymbol{a}, \boldsymbol{b}$ defining bounds $[[\boldsymbol{a}, \boldsymbol{b}]] \subseteq \operatorname{dom} f$, and a vector of initial points $(\boldsymbol{x}_i, y_i)_i$ satisfying $y_i = f(\boldsymbol{x}_i)$ for all i; Output: an initialized Bayesian optimizer for f with the posterior probability distribution on f updated using the available samples $(\boldsymbol{x}_i, y_i)_i$.

SUGGEST Inputs: none. Output: A maximizer $\boldsymbol{z} \in [[\boldsymbol{a}, \boldsymbol{b}]]$ of the acquisition function over $\boldsymbol{x}$, where the acquisition function is computed using the current posterior distribution.

OBSERVE Inputs: $(\boldsymbol{x}, y)$. Output: the Bayesian optimizer with updated posterior probability based on $y = f(\boldsymbol{x})$.

Algorithm 1 Basic optimization algorithm using the BO solver $A^{\max}$ for maximizing $f(\boldsymbol{x})$.

$MaxIter$ – a bound on the number of sampling iterations
Sample f at m input points based on a heuristic:
Let $\boldsymbol{x}_i \in [[\boldsymbol{a}, \boldsymbol{b}]]$ and $y_i = f(\boldsymbol{x}_i)$ for $i = 1, \ldots, m$
$A \leftarrow A^{\max}.\mathrm{INIT}(\boldsymbol{a}, \boldsymbol{b}, (\boldsymbol{x}_i, y_i)_i)$
for $j = m + 1, \ldots, MaxIter$ **do**
  $\boldsymbol{x}_j \leftarrow A.\mathrm{SUGGEST}$
  Sample $y_j = f(\boldsymbol{x}_j)$
  $A \leftarrow A.\mathrm{OBSERVE}(\boldsymbol{x}_j, y_j)$
end
$y_k \leftarrow \max\{y_1, \ldots, y_{MaxIter}\}$
return a near-optimal solution: $(\boldsymbol{x}_k, y_k)$

We will also use a minimizing BO solver $B^{\min}$ that has the same interface. This basic BO algorithm does not take as input any constraints on the inputs of f thus it is not concerned with feasibility of the computed near-optimal point. Furthermore, this algorithm is not concerned with robustness of the computed near-optimal point either and does not give any estimates of how far from the real maximum the computed near-optimal solution is. We use this basic BO algorithm, complemented with an SMT procedure where needed, as a building block of a more comprehensive, hybrid optimization algorithm that gives formal guarantees that the computed near-optimal points are feasible, stable, with values close to the real maximum up to arbitrary given accuracy. We will define stability and accuracy formally in the next section.

4 Stability and Accuracy

Given a real-valued function f on a bounded space $X \subset \mathbb{R}^n$, we are interested in regions of X defined by the *stability guard* θ which we assume is a reflexive binary relation over $\operatorname{dom} f$. In particular, we address the problem of maximizing the minimum value of f in these regions, as stated formally in Definition 1.

The region defined by $\theta(\boldsymbol{x}, \cdot)$ could for example be a ball with a fixed radius around $\boldsymbol{x} \in X$ or a radius relative to $\|\boldsymbol{x}\|$. It could also take a more complicated shape and we do not impose any restrictions on it except that of being definable by a quantifier-free formula θ. In the following, we assume f is in $\mathcal{F}_D$, i.e., defined by a formula $F(x_1, \ldots, x_n, y)$. We also assume that X can be defined by safety constraints.

Define the minimal value of f in the θ region of $\boldsymbol{x} \in X$ as

$$\llbracket \boldsymbol{x} \rrbracket_{\theta,f} := \min_{\substack{\boldsymbol{x}' \in X \\ \theta(\boldsymbol{x},\boldsymbol{x}')}} f(\boldsymbol{x}').$$

Definition 1. Given a stability guard θ for a definable and continuous $f : \mathbb{R}^n \to \mathbb{R}$ and a compact set $X \subseteq \operatorname{dom} f$, the problem

$$\max_{\boldsymbol{x} \in X} \min_{\substack{\boldsymbol{x}' \in X \\ \theta(\boldsymbol{x},\boldsymbol{x}')}} f(\boldsymbol{x}') \tag{1}$$

is called *max-min optimization under stability guard* θ. Consider $\varepsilon > 0$. We refer to $\tilde{\boldsymbol{x}} \in X$ as a solution to this problem with *accuracy* ε, or *ε-solution*, if $\tilde{y} \leq y^* < \tilde{y} + \varepsilon$ holds where $y^* = \llbracket \boldsymbol{x}^* \rrbracket_{\theta,f}$ for a solution $\boldsymbol{x}^*$ to (1), and $\tilde{y} = \llbracket \tilde{\boldsymbol{x}} \rrbracket_{\theta,f}$.

In a nutshell, the value $\tilde{y}$ in the definition of an ε-solution $\tilde{\boldsymbol{x}}$ to the optimization problem under a stability guard θ is an approximation of the exact solution with guaranteed accuracy ε. It provides a lower bound for all the function's values in the stability region around $\tilde{\boldsymbol{x}}$ that is no further away from the maximum value y^* than ε. When f, X and θ are clear from the context, we will call $\tilde{\boldsymbol{x}}$ just a *(θ-)stable ε-solution* for short. In the next section, we present algorithms based on combination of BO and SMT that solve this problem.

5 Optimisation Procedure

The problem (1) is equivalent to:

$$\max y \text{ s.t. } G(y) \tag{2}$$

where

$$G(y) = \exists \boldsymbol{x}\, (\forall \boldsymbol{x}' \forall y'\, (\theta(\boldsymbol{x}, \boldsymbol{x}') \land F(\boldsymbol{x}', y') \to y \leq y')). \tag{3}$$

Let T denote a candidate bound on the value y^* of a solution $\boldsymbol{x}^*$ to (1). Formulas of the form $G(T)$ with a fixed value T are also known to be in the GEAR-fragment of $\exists^*\forall^*$ formulas [Brauße *et al.*, 2020]. In the case we have safety constraints we can conjunctively adjoin them to the condition $y \leq y'$ in Equation (3). Such additional constraints do not affect our algorithms and we will omit them for clarity of the exposition.

We first recall the decision procedure GEARSAT$_\delta$ [Brauße *et al.*, 2020] shown in Algorithm 2, which we enhance with BO solvers below. It takes a potential bound T and either verifies T to be a lower bound on the optimum y^* or proves it to be an upper bound on y^*. GEARSAT$_\delta$ alternates two phases: search for candidate solution and search for counter-example for stability around the candidate solution. It does that by first finding a point $\boldsymbol{x}$ for which $f(\boldsymbol{x}) \geq T$ holds – a *candidate for a stable, safe, and accurate solution*. If there is none, clearly T is an upper bound on y^*. Otherwise, it checks whether when $\boldsymbol{x}$ is seen as the center of the stability region $\theta(\boldsymbol{x}, \cdot)$, there is a *counter-example* $\boldsymbol{x}'$, that is, $\theta(\boldsymbol{x}, \boldsymbol{x}')$ holds but $f(\boldsymbol{x}') \geq T$ does not (represented by $D_i(\boldsymbol{x}', y')$ constraint in Algorithm 2). In the case when there are no counter-examples with this property, we can be sure that $\theta(\boldsymbol{x}, \boldsymbol{x}')$ implies $f(\boldsymbol{x}') \geq T$ for all $\boldsymbol{x}' \in X$. Otherwise, $\boldsymbol{x}'$ is a counter-example and the algorithm excludes the region $\theta(\cdot, \boldsymbol{x}')$ around it from the search for the next candidate for the same bound T. The stability condition θ guides the proof search by generating lemmas excluding regions around counter-examples.

Here, $\delta \geq 0$ refers to a constant which is used to ensure that solutions have a distance of at least δ from regions defined by θ containing counter-examples for bound T. This is done by learning lemmas of the form $\neg\theta_\delta(\boldsymbol{x}, \boldsymbol{d})$ for each counter-example $\boldsymbol{d}$, where θ_δ is the δ-relaxation of θ defined by

$$\exists \boldsymbol{z}\, (\|\boldsymbol{x} - \boldsymbol{z}\| \leq \delta \land \theta(\boldsymbol{z}, \boldsymbol{d})).$$

Termination and δ-completeness of GEARSAT$_\delta$ was shown in [Brauße *et al.*, 2020]. We can use GEARSAT$_\delta$ to find an optimal value with accuracy ε by a binary search of lower and upper bounds on the optimal value of the objective function until $T \leq y^* < T + \varepsilon$. Then $\tilde{y} = T$ is the minimal value $\llbracket \tilde{\boldsymbol{x}} \rrbracket_{\theta,f}$ of an ε-solution $\tilde{\boldsymbol{x}}$.

The key search parts in GEARSAT$_\delta$ rely on finding points (either candidates or counter-examples). One way to achieve this is by using an SMT solver to find points satisfying corresponding constraints (as done in [Brauße *et al.*, 2020]) but this can be computationally expensive. In this work we propose to delegate the search part to BO and the certification part to SMT checks. In this way we take the best from both worlds: efficient search in complex spaces from BO and formal guarantees on the optimization results from SMT.

Algorithm 2 (GEARSAT$_\delta$) General procedure for deciding whether a constant T is a lower or upper bound on the solution y^* of (2).

```
N ← ∅        ▷ known counter-examples and upper bounds
F_1(x, y) ← F(x, y)
for i = 1, 2, 3, ... do
    C_i(x, y) ← F_i(x, y) ∧ y ≥ T
    if C_i(x, y) is unsat then return upper fi
    (c_i, y_i) ← solution of C_i(x, y)
    D_i(x', y') ← θ(c_i, x') ∧ F_i(x', y') ∧ y' < T
    if D_i(x', y') is unsat then return lower fi
    (d_i, z_i) ← solution of D_i(x', y')
    N ← N ∪ {(d_i, z_i)}
    F_{i+1}(x, y) ← F_i(x, y) ∧ ¬θ_δ(x, d_i)
end
```

Our algorithms do not depend on particular types of BO and SMT solvers used. We only assume that the SMT solver supports quantifier-free fragment including formulas F and θ.

First we integrate BO into GEARSAT$_\delta$ for searching counter-examples. We note that whenever $\theta_\delta(\boldsymbol{c}_i, \boldsymbol{x}')$ is equivalent to membership in the Cartesian product $[[\boldsymbol{a}_i, \boldsymbol{b}_i]]$ (as is always the case in our application), BO lends itself well to implement the search for a counter-example by solving $D_i(\boldsymbol{x}', y)$, as is shown in Algorithm 3. This works by starting a new minimizing BO search in $[[\boldsymbol{a}_i, \boldsymbol{b}_i]]$ once Algorithm 2 found a candidate $\boldsymbol{c}_i$. Note that variables $N, P, \boldsymbol{c}_i, \boldsymbol{d}_i$ and T are global across all algorithms in this section.

Algorithm 3 Finding counter-examples: Solving $D_i(\boldsymbol{x}', y')$ with Bayesian optimization and SMT in the i-th iteration of Algorithm 2.

```
Let (a_i, b_i) denote the bounds on x' implied by θ(c_i, x')
in D_i(x', y') and let T denote the bound on y' in D_i(x', y')
Let e_1, ..., e_k ∈ [[a_i, b_i]]                          ▷ BO
if f(e_j) < T for some j ∈ {1, ..., k} then
    d_i ← e_j
    return (d_i, f(d_i))
else
    B_i ← B^min.INIT(a_i, b_i, (e_j, f(e_j))_j)
    for j = 1, ..., MaxIter do
        d_i ← B_i.SUGGEST
        z_i ← f(d_i)
        B_i ← B_i.OBSERVE(d_i, z_i)
        if z_i < T then return (d_i, z_i) fi
    end
    if D_i(x', y') is unsat then return unsat fi         ▷ SMT
    return (d_i, z_i) ← solution of D_i(x', y')
fi
```

Next, we integrate BO for finding candidate solutions. For this we need to guide BO to generate points outside of regions excluded by generated lemmas. Even though most BO solvers do not support constraints like the lemmas $\neg\theta_\delta(\boldsymbol{x}, \boldsymbol{d}_m)$ learned by Algorithm 2, when $X \subseteq \operatorname{dom} f$ is equivalent to membership in $[[\boldsymbol{a}, \boldsymbol{b}]]$ for some $\boldsymbol{a}, \boldsymbol{b}$, it is still possible to use them in the search for a candidate $\boldsymbol{c}$ in our setting. We achieve this by penalizing any suggestion $\boldsymbol{c}_i$ that satisfies $\theta_\delta(\boldsymbol{x}, \boldsymbol{d})$, for any counter-example $\boldsymbol{d}$, with the minimal value of a counter-example found when generating the lemma. This is shown in Algorithm 4. Alongside the SMT solver it maintains the maximizing BO solver A_i that is initialized externally with selected points which are either generated in previous iterations or randomly.

Note that due to penalizing any of A_i's suggestions inside regions containing counter-examples, the function that A_i observes is not necessarily f. Whenever A_i's suggestion $\boldsymbol{c}_i$ lies in a region around a previous counter-example $\boldsymbol{d}$ in the set of known ones N, that is, $\theta(\boldsymbol{c}_i, \boldsymbol{d})$ holds, we make A_i believe that the value of the function it optimizes is the minimum value of all counter-examples around $\boldsymbol{c}_i$ instead of $f(\boldsymbol{c}_i)$. Since $\boldsymbol{d}$ was a counter-example, specifically $f(\boldsymbol{d}) < T$, this has the effect of penalizing the suggestion $\boldsymbol{c}_i$ since $\boldsymbol{d}$ has already been proved to be a counter-example in the region defined by $\theta(\boldsymbol{c}_i, \cdot)$.

Algorithm 4 Finding candidates: Solving $C_i(\boldsymbol{x}, y)$ with Bayesian optimization and SMT in the i-th iteration of Algorithm 2. A_i maximizes.

```
if i > 1 then                    ▷ record previous counter-example
    A_i ← A_{i-1}.OBSERVE(c_{i-1}, f(d_{i-1}))
fi
for j = 1, ..., MaxIter do                                    ▷ BO
    c_i ← A_i.SUGGEST
    U ← {(d, z) ∈ N : θ(c_i, d) ∧ z < T}
    if U ≠ ∅ then                   ▷ c_i is excluded by lemmas
        y_i ← min{z : (d, z) ∈ U}
    else                          ▷ c_i is an eligible candidate
        y_i ← f(c_i)
        if y_i ≥ T then return (c_i, y_i) fi          ▷ f(c_i) ≥ T
        N ← N ∪ {(c_i, y_i)}
    fi
    A_i ← A_i.OBSERVE(c_i, y_i)                           ▷ y_i < T
end
if C_i(x, y) is unsat then return unsat fi                   ▷ SMT
return (c_i, y_i) ← solution of C_i(x, y)
```

We call GEARSAT$_\delta$ with integrated BO for counter-examples and candidate search GEAROPT$_\delta$-BO. This procedure is shown in Algorithm 5. The following theorem states the correctness of GEAROPT$_\delta$-BO.

Theorem 1. *For any accuracy $\varepsilon > 0$, any stability guard θ and $\delta > 0$, GEAROPT$_\delta$-BO is a sound, δ-complete and terminating procedure for the problem of finding safe, optimal and θ-stable ε-solutions.*

Proof. (Sketch) The proof follows from the properties of GEARSAT$_\delta$ (Algorithm 2) which was shown to be a sound and δ-complete decision procedure for finding safe and stable solutions [Brauße *et al.*, 2020]. δ-completeness means that if GEARSAT$_\delta$ terminates with unsat, then around any potential solution there is a counter-example in its θ_δ region. By the δ-completeness, the binary search performed by Algorithm 5 terminates with a solution to the max-min optimization problem. Since all BO enhancements are complemented by SMT checks these can be seen as an additional guidance that does not affect the correctness properties of the algorithm. □

One of the distinguishing features of GEAROPT$_\delta$-BO is the exchange of candidates and counter-examples between BO and SMT solvers. In particular, when BO finds a counter-example (Algorithm 3), the generated symbolic lemma is used (a) by the SMT solver to exclude regions around this counterexample (Algorithm 2), and (b) in BO for penalizing points in excluded regions (Algorithm 4). In the other direction, candidate solutions found by the SMT solver are used to initialize the BO solver (update of P in Algorithm 5). Also, failed attempts of the BO solver to find candidates are recorded (update of N in Algorithm 4) in order to steer it away from unstable regions. As shown in Section 6, this exchange of candidates and counter-examples between BO and SMT solvers proved to be critical for both achieving tighter bounds and making runtimes feasible in our experiments.

Algorithm 5 (GEAROPT$_\delta$-BO) General procedure for optimizing $f : \mathbb{R}^n \to \mathbb{R}$ represented by predicate F under stability conditions θ. Arbitrary candidate lower and upper bounds l_0, u_0 such that $l_0 < u_0$.

$P \leftarrow \varnothing$ ▷ known candidates and lower bounds
$N \leftarrow \varnothing$ ▷ known counter-examples and upper bounds
$l \leftarrow -\infty$ ▷ lower bound
$u \leftarrow +\infty$ ▷ upper bound
loop
  if $u = +\infty$ **then** ▷ binary search for upper bound
    $(T, u_0) \leftarrow (u_0, 2u_0 - l_0)$
  else if $l = -\infty$ **then** ▷ binary search for lower bound
    $(T, l_0) \leftarrow (l_0, 2l_0 - u_0)$
  else ▷ binary search for optimum
    $T \leftarrow (l + u)/2$
  fi
  $F_1(\boldsymbol{x}, y) \leftarrow F(\boldsymbol{x}, y)$
  $A_1 \leftarrow A^{\max}.\textsc{Init}(\boldsymbol{a}, \boldsymbol{b}, P \cup N)$
  for $i = 1, 2, 3, \ldots$ **do**
    $C_i(\boldsymbol{x}, y) \leftarrow F_i(\boldsymbol{x}, y) \wedge y \geq T$
    if Algorithm 4 returns unsat on $C_i(\boldsymbol{x}, y)$ **then**
      $u \leftarrow T$
      break
    fi
    $(\boldsymbol{c}_i, y_i) \leftarrow$ solution of $C_i(\boldsymbol{x}, y)$
    $D_i(\boldsymbol{x}', y') \leftarrow \theta(\boldsymbol{c}_i, \boldsymbol{x}') \wedge F_i(\boldsymbol{x}', y') \wedge y' < T$
    if Algorithm 3 returns unsat on $D_i(\boldsymbol{x}', y')$ **then**
      $l \leftarrow T$ ▷ $\boldsymbol{c}_i$ is solution to bound T
      $P \leftarrow P \cup \{(\boldsymbol{c}_i, y_i)\}$
      break
    fi
    $(\boldsymbol{d}_i, z_i) \leftarrow$ solution of $D_i(\boldsymbol{x}', y')$
    $N \leftarrow N \cup \{(\boldsymbol{d}_i, z_i)\}$ ▷ $\boldsymbol{d}_i$ is a counter-example
    $F_{i+1}(\boldsymbol{x}, y) \leftarrow F_i(\boldsymbol{x}, y) \wedge \neg\theta_\delta(\boldsymbol{x}, \boldsymbol{d}_i)$
  end
  if $l + \varepsilon > u$ **then break fi** ▷ false if l or u is $\pm\infty$
end loop
return $\{(\boldsymbol{x}, l) : (\boldsymbol{x}, y) \in P \wedge l \leq y\}$

6 Benchmarks

GEAROPT$_\delta$-BO is implemented in the solver called SMLP (https://github.com/fbrausse/smlp). As Bayesian optimizers $A^{\max}, B^{\min}$, we used the SKOPT implementation [Pedregosa *et al.*, 2011] based on Gaussian processes, with acquisition function gp_hedge. The SMT part is implemented using the state of the art solver Z3 [de Moura and Bjørner, 2008]. Just like the implementation of GEARSAT$_\delta$ in [Brauße *et al.*, 2020], besides the constraints associated with neural networks, our implementation of GEAROPT$_\delta$-BO in SMLP supports constraints over reals, integers and finite sets; this enables SMLP to optimize systems with numerical and categorical variables. Furthermore, SMLP supports optimization of $f(x_1, \ldots, x_n)$ by optimizing over only part of the variables $x_1, \ldots, x_k$ while the remaining variables $x_{k+1}, \ldots, x_n$ are treated as free inputs; thus safety, accuracy and stability of the selected near-optimal solutions for $x_1, \ldots, x_k$ are guaranteed for all legal input combinations of $x_{k+1}, \ldots, x_n$

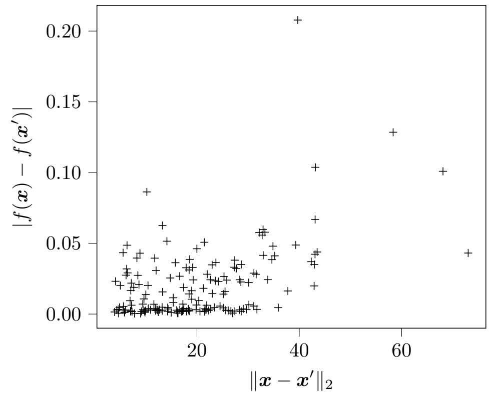

Figure 1: Correspondence between bound on optimum and stability radius $\|\boldsymbol{x} - \boldsymbol{x}'\|_2$ achievable on solutions found by a purely BO-based approach on instance 3:1:1.

as defined by the safety constraints. Transformation of GEAR formulas enabling this is explained in [Brauße *et al.*, 2020, Section V].

Let us note that there is a tradeoff between the size of the stability regions and proximity of the value of the solution to the pointwise optimum. One can iteratively shrink the stability regions to get arbitrary close to the pointwise optimum.

We evaluated GEAROPT$_\delta$-BO on 6 industrial examples coming from the Electrical Validation Lab at Intel. These are neural network models representing signal integrity of transmitters and receivers of a channel to a peripheral device. This application requires solutions to be safe and stable (see, Section 4), moreover the radii of stability regions are required to be proportional to the value of their respective centers. We evaluated all 4 combinations with and without BO-guided searches for candidates and counter-examples, respectively. The accuracy was set to $\varepsilon = 0.05$. These results are shown in Table 1. In the left-most column i refers to the problem instance, c to whether BO search was used for candidates and d to whether BO search was used for counter-examples.

6.1 Optimality under Stability Guards

Throughout our experiments, the combination of BO with SMT solvers proved to find the best bound $\tilde{y} = 1$ to the optimum, whereas SMT alone timeouts in many cases. This can be attributed to the facts that 1) when BO failed to find counter-examples (n_{sa}), none did exist (N_{sa}), and that 2) the average time taken to find a counter-example is much shorter for BO ($t_{\mathrm{ce}}/n_{\mathrm{ce}}$) than for SMT ($T_{\mathrm{ce}}/N_{\mathrm{ce}}$ for $N_{\mathrm{ce}} \neq 0$ results). This suggests that BO constitutes a very good heuristic for finding counter-examples. On the other hand, there are many failed candidates suggested by BO (n_{cai} in *:1:*) which indicates that BO alone is not able to find stable candidates and SMT is required to guarantee stability of the solutions.

The combinations *:0:1 correspond to those where BO tries to refute stability of SMT candidates. Throughout, it manages to do that with on average $n_{\mathrm{cci}}/n_{\mathrm{ce}} < 3$ iterations in Algorithm 3. On the other hand, for *:1:1 when BO tries to find counter-examples to BO candidates, this average with value ≈ 8.8 is much higher on average in our experiments.

i:c:d	bound T	N_{cap}	N_{ce}	N_{sa}	T_{cap}	T_{ce}	T_{sa}	n_{cai}	n_{cci}	n_{cap}	n_{can}	n_{ce}	n_{sa}	n_{un}	t_{cap}	t_{can}	t_{ce}	t_{sa}	time
0:0:0	≥ 0.80	1376	1375	1	55114.7	54757.5	20.3	0	0	0	0	0	0	0	0.0	0.0	0.0	0.0	> 2d
0:0:1	≥ 0.80	5049	0	1	105017.1	0.0	21.2	0	6711	0	0	5048	1	0	0.0	0.0	829.8	81.3	> 2d
0:1:0	≥ 0.95	19	30	2	471.6	92.0	5.3	1114	0	13	19	0	0	0	379.1	105059.2	0.0	0.0	> 2d
0:1:1	≥ 0.95	3	0	2	233.3	0.0	9.2	1113	1333	254	3	255	2	0	69931.5	19954.4	248.4	58.1	> 2d
1:0:0	≥ 0.80	3254	3253	1	80374.5	30002.8	18.8	0	0	0	0	0	0	0	0.0	0.0	0.0	0.0	> 2d
1:0:1	≥ 0.50	8094	0	0	34519.6	0.0	0.0	0	16674	0	0	8094	0	0	0.0	0.0	835.5	0.0	> 2d
1:1:0	≥ 1.00	5	111	3	389.0	332.5	31.9	1163	0	109	5	0	0	0	61212.0	43901.3	0.0	0.0	> 2d
1:1:1	1.00	2	0	4	322.2	0.0	29.2	525	423	74	2	72	4	0	6125.5	859.5	33.5	330.6	8353
2:0:0	≥ 0.95	1025	1023	2	31240.7	84147.1	87.8	0	0	0	0	0	0	0	0.0	0.0	0.0	0.0	> 2d
2:0:1	≥ 0.50	5000	0	0	36717.1	0.0	0.0	0	6420	0	0	5001	0	0	0.0	0.0	349.4	0.0	> 2d
2:1:0	≥ 1.00	21	31	3	1165.4	195.1	27.1	1239	0	13	21	0	0	0	364.9	101490.2	0.0	0.0	> 2d
2:1:1	≥ 1.00	12	0	3	1153.9	0.0	22.0	1247	359	60	12	69	3	0	16773.8	67275.1	64.4	180.3	> 2d
3:0:0	≥ 0.50	2608	2608	0	70288.7	42255.8	0.0	0	0	0	0	0	0	0	0.0	0.0	0.0	0.0	> 2d
3:0:1	≥ 0.95	9301	1154	2	23559.8	12381.9	82.9	0	21674	0	0	8145	2	1154	0.0	0.0	1135.9	111.4	> 2d
3:1:0	1.00	0	16	4	0.0	93.7	33.9	110	0	20	0	0	0	0	201.3	0.0	0.0	0.0	399
3:1:1	1.00	0	0	4	0.0	0.0	21.6	96	365	22	0	18	4	0	118.0	0.0	42.6	281.6	514
4:0:0	≥ 0.80	1112	1110	1	8193.1	107511.3	37.0	0	0	0	0	0	0	0	0.0	0.0	0.0	0.0	> 2d
4:0:1	≥ 0.80	1406	0	1	104095.2	0.0	44.3	0	3426	0	0	1405	1	0	0.0	0.0	3168.8	138.6	> 2d
4:1:0	≥ 0.95	4	136	2	1022.2	498.7	12.4	1098	0	134	4	0	0	0	70635.3	27534.9	0.0	0.0	> 2d
4:1:1	≥ 0.95	6	0	2	917.3	0.0	6.4	1143	221	64	6	68	2	0	48538.4	39695.9	10.1	118.4	> 2d
5:0:0	≥ 0.50	1446	1446	0	42628.9	70583.7	0.0	0	0	0	0	0	0	0	0.0	0.0	0.0	0.0	> 2d
5:0:1	≥ 0.50	14725	388	0	95689.8	1709.6	0.0	0	27543	0	0	14337	0	388	0.0	0.0	1077.3	0.0	> 2d
5:1:0	1.00	0	29	4	0.0	131.8	27.5	209	0	33	0	0	0	0	852.7	0.0	0.0	0.0	1195
5:1:1	1.00	0	0	4	0.0	0.0	25.7	161	382	33	0	29	4	0	478.4	0.0	29.5	414.4	1107

Table 1: Indicators of SMLP for all combinations of en-/disabled BO heuristics for candidates and counter-examples. T: obtained bound on optimal value. N_*/n_*: number of SMT/BO solutions. T_*/t_*: time of SMT/BO solutions. $\mathrm{ca(p|n)}$: $\exists$ or $\neg\exists$ candidate. $\mathrm{ce|sa}$: $\exists$ or $\neg\exists$ counter-example. $\mathrm{cai|cci}$: total BO calls for candidate or counter-example. un: BO library failure.

This suggests that the quality of BO candidates when guided by BO counter-examples (*:1:1) compares favorably to that of the SMT candidates with BO counter-examples (*:0:1).

6.2 Stability

Figure 1 shows the dependencies between the stability radius and the bound on the optimum achievable on candidate points. We can observe that initial candidates $\boldsymbol{x}$ found by BO are not necessarily stable ($\boldsymbol{x}'$ close to $\boldsymbol{x}$, but objective values differ considerably wrt. the accuracy ε), and therefore usage of SMT is essential to discover stable solutions.

6.3 Performance

From the timings in Table 1 we can see that the best bounds and timings are achieved when SMT is combined with BO for both candidate and counter-example search. When SMT is combined with BO time spent in the SMT solver (T_*) is considerably reduced in many cases by two orders of magnitude. On the other hand, the time spent by BO is also considerable increases in these cases. It turns out that as long as the total number of candidates found by BO-solver A remains low ($n_{\mathrm{cap}} < 50$), the asymptotically cubic complexity of the Gaussian process appears to be negligible compared to the overhead of rigorously solving the existential candidate formula by SMT. On the other hand, this advantage vanishes quickly after that, which motivates employing a kind of restart process similar to that successfully practiced by current SAT and SMT solvers – at least for the BO solver A. Initial experiments suggest that training samples for the restarted BO solver require careful selection.

All in all, we can see BO and SMT complement each other to solve the problem stated in Definition 1. BO's ability to rapidly produce candidates and counter-examples initially allows the combination to proceed to optimal regions quickly in most cases while still providing the formal guarantees on the validity and accuracy of stable solutions.

7 Conclusions

We introduced a hybrid optimization algorithm GEAROPT$_\delta$-BO that uses Bayesian optimization and SMT solvers as its building blocks. SMT solving is used to establish formal guarantees to optimality and stability of the computed solutions; while BO is used to suggest valuable candidates towards stable near-optimal solutions and significantly speeds up the overall search. In this way we combine the strengths of both approaches: the power of statistical inference by BO to guide the search with formal guarantees provided by SMT.

To the best of our knowledge this is the first work that combines BO and SMT solving to overcome basic limitations of the BO, in particular, its inability to give formal guarantees of stability and accuracy of the computed optimum, which becomes possible to resolve in cases when the objective function is given explicitly rather than as a black-box.

We believe that the observation that BO is very good in finding counter-examples in large multi-dimensional spaces, opens up new opportunities for applying BO for counter-example generation and directing the search in multiple areas of automated reasoning and formal verification.

Acknowledgements

This research was supported by a grant from Intel Corporation.

References

[Beyer and Sendhoff, 2007] Hans-Georg Beyer and Bernhard Sendhoff. Robust optimization – a comprehensive survey. *Computer Methods in Applied Mechanics and Engineering*, 196(33):3190–3218, 2007.

[Bogunovic *et al.*, 2018] Ilija Bogunovic, Jonathan Scarlett, Stefanie Jegelka, and Volkan Cevher. Adversarially robust optimization with Gaussian processes. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: NeurIPS 2018*, pages 5765–5775, 2018.

[Brauße *et al.*, 2019] Franz Brauße, Konstantin Korovin, Margarita V. Korovina, and Norbert Th. Müller. A CDCL-style calculus for solving non-linear constraints. In Andreas Herzig and Andrei Popescu, editors, *Frontiers of Combining Systems - 12th International Symposium, FroCoS 2019, London, UK. Proceedings*, volume 11715 of *LNCS*, pages 131–148. Springer, 2019.

[Brauße *et al.*, 2020] Franz Brauße, Zurab Khasidashvili, and Konstantin Korovin. Selecting stable safe configurations for systems modelled by neural networks with ReLU activation. In *2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel*, pages 119–127. IEEE, 2020.

[Brauße *et al.*, 2021] Franz Brauße, Konstantin Korovin, Margarita V. Korovina, and Norbert Th. Müller. The ksmt calculus is a δ-complete decision procedure for non-linear constraints. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings*, volume 12699 of *LNCS*, pages 113–130. Springer, 2021.

[Cardelli *et al.*, 2019] Luca Cardelli, Marta Kwiatkowska, Luca Laurenti, and Andrea Patane. Robustness guarantees for Bayesian inference with Gaussian processes. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA*, pages 7759–7768. AAAI Press, 2019.

[Cimatti *et al.*, 2013] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. The MathSAT5 SMT solver. In Nir Piterman and Scott A. Smolka, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013. Proceedings*, volume 7795 of *LNCS*, pages 93–107. Springer, 2013.

[de Moura and Bjørner, 2008] Leonardo de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary. Proceedings*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[Frazier, 2018] Peter I. Frazier. A tutorial on Bayesian optimization. *CoRR*, abs/1807.02811, 2018.

[Gao *et al.*, 2012] Sicun Gao, Jeremy Avigad, and Edmund M. Clarke. δ-complete decision procedures for satisfiability over the reals. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Proceedings*, volume 7364 of *LNCS*, pages 286–300. Springer, 2012.

[Gardner *et al.*, 2014] Jacob R. Gardner, Matt J. Kusner, Zhixiang Eddie Xu, Kilian Q. Weinberger, and John P. Cunningham. Bayesian optimization with inequality constraints. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 937–945. JMLR.org, 2014.

[Mockus, 1975] Jonas Mockus. On Bayesian methods for seeking the extremum. *Optimization Techniques IFIP Technical Conference*, 1975.

[Mockus, 1989] Jonas Mockus. Bayesian approach to global optimization: Theory and applications. *Kluwer Academic Publishers*, 1989.

[Pedregosa *et al.*, 2011] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Rasmussen and Williams, 2006] Carl Edward Rasmussen and Christopher K. I. Williams. Gaussian processes for machine learning. *The MIT Press, Cambridge, MA*, 2006.

[Sanders *et al.*, 2019] Nicholas D. Sanders, Richard M. Everson, Jonathan E. Fieldsend, and Alma A. M. Rahat. A Bayesian approach for the robust optimisation of expensive-to-evaluate functions. *IEEE Transactions on Evolutionary Computing. arXiv : 1904.11416v2*, pages 2951–2959, 2019.

[Snoek *et al.*, 2012] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.