Fast Design Space Exploration of Nonlinear Systems: Part I
Sanjai Narain, Emily Mak, Dana Chee, Brendan Englot, Senior Member, IEEE,
Kishore Pochiraju, Niraj K. Jha, Fellow, IEEE, and Karthik Narayan

Abstract—System design tools are often only available as input–output blackboxes: for a given design as input, they compute an output representing system behavior. Blackboxes are intended to be run in the forward direction. This article presents a new method of solving the “inverse design problem,” namely, given requirements or constraints on output, find an input that also optimizes an objective function. This problem is challenging for several reasons. First, blackboxes are not designed to be run in reverse. Second, inputs and outputs can be discrete and continuous. Third, finding designs concurrently satisfying a set of requirements is hard because designs satisfying individual requirements may conflict with each other. Fourth, blackbox evaluations can be expensive. Finally, evaluations can sometimes fail to produce an output due to nonconvergence of underlying numerical algorithms. This article presents CNMA, a new method of solving the inverse problem that overcomes these challenges. CNMA tries to sample only the part of the design space relevant to solving the inverse problem, leveraging the power of neural networks, mixed-integer linear programs, and a new learning-from-failure feedback loop. This article also presents a parallel version of CNMA that improves the efficiency and quality of solutions over the sequential version and tries to steer it away from local optima. CNMA’s performance is evaluated against conventional optimization methods for seven nonlinear design problems of 8 (two problems), 10, 15, 36 and 60 real-valued dimensions and one with 186 binary dimensions. Conventional methods evaluated are stable, off-the-shelf implementations of the Bayesian optimization with the Gaussian Processes, Nelder–Mead, and Random Search. The first two do not even produce a solution for problems that are high dimensional, having both discrete and continuous variables or whose blackboxes fail to return values for some inputs. CNMA produces solutions for all problems. When conventional methods do produce solutions, CNMA improves upon their performance by 1%–87%.

Index Terms—Blackbox optimization, constrained optimization, mixed-integer linear program (MILP), neural networks (NNs), optimization, sample efficiency.

I. INTRODUCTION

SYSTEM design knowledge is often encapsulated inside blackboxes such as simulators, spreadsheets and program scripts. Blackboxes are, typically, nonlinear functions that accept a design as input and produce a representation of system behavior as output. New designs can be created by solving the “inverse problem”: from requirements or constraints on blackbox output, compute an input that also optimizes an objective function.

Relevant to solving this problem are the mature, constrained nonlinear optimization methods [1]–[10]. Also available is a large online collection of these methods [11] along with a companion guide [12]. These methods can be adapted to solve the problem of finding $x$ that optimizes an objective function $\phi(x, F(x))$ subject to a constraint $P(x, F(x))$ where $F$ is a blackbox function, $\phi$ is an objective function, and $P$ is a constraint or a requirement. Any solution $x$ is a solution to the inverse problem since $x$ is an input to $F$ that optimizes $\phi$ and satisfies $P$ on the output of $F$.

Fig. 1 presents a taxonomy of these methods. These fall into two categories: 1) derivative based and 2) derivative-free. The former compute derivatives of the objective function to determine the direction in which to search for a point where the derivative becomes 0. They are restricted to smooth, continuous functions. Thus, they do not apply to functions with discontinuities or discrete variables. Moreover, derivative computation is not sample efficient in that it requires a large number of function evaluations. Thus, these methods are infeasible when function evaluation is expensive, as is often the case for blackboxes.

Derivative-free methods [5], [7] try to overcome the limitations of derivative-free ones. One class of such methods is called direct search whose well-known members include Nelder–Mead (NM) [6] and COBYLA [13]. They maintain a simplex (convex hull) of points around the current point and use it to compute the next point to sample in the direction of the optima. These methods require starting points whose incorrect choices can cause the methods to be stuck in local optima.

Another class of derivative-free methods is metamodeling or surrogate based [7], [8]. They do not require starting points. Instead, they sample the blackbox function at some set of points and construct a surrogate model by fitting the values to a mathematical model using machine learning. Active-learning metamodeling methods are conservative in the number of samples they evaluate. They do this by constructing a merit or acquisition function from the surrogate model. This merit function is optimized to compute the best point to sample next. If this point does not satisfy a halting condition,
the point is added to the set of samples and the search is restarted. Examples include ALAMO [3] and Bayesian optimization (BO) with Gaussian Processes (BO/GP) [9]. In geostatistics, BO/GP is called Kriging [1] and the use of Kriging for the circuit design is reported in [14]. In earlier BO/GP versions, the complexity of building surrogate functions was cubic in the number of samples [1] although asymptotically faster versions on GPUs are reported in [15]. To optimize with constraints, various extensions to BO/GP have been presented in [16]–[19]. Parallel BO/GP algorithms have been presented in [20]–[22].

Nonactive-learning metamodeling methods, e.g., [23]–[26], directly use the surrogate as a fast evaluator of the blackbox function for use by optimization methods, including the derivative-based ones. They may not be feasible for higher dimensional functions where the number of samples needed to construct an accurate enough surrogate may be astronomical.

Genetic algorithms [27] form a third type of derivative-free optimization. A population of samples is maintained that is systematically improved over multiple generations. While such algorithms can avoid local optima and operate on functions with discrete and continuous variables, they may converge to the final solutions slowly. The sister Part II of this article [28] shows how the conjunction of genetic algorithm and CNMA can overcome this problem.

The above methods can be susceptible to failure of objective functions to evaluate, as can happen, for example, due to non-convergence of computational fluid dynamics simulators [29]. If an artificial value has to be assigned to the function it could distort the shape of the new objective function and CNMA can overcome this problem.

Many of the above methods handle constraints indirectly by reducing constrained optimization with a sequence of one or more unconstrained optimization problems [2], [4], [30], [31]. The principle is to encode the cost of violating a constraint as a penalty/barrier function and rely on an optimization engine to solve for the inverse problem for blackboxes. Formally, CNMA finds values of \( x \) and \( y \) that optimize \( \phi(x, y) \) such that \( F(x) = y \land P(x, y) \) where \( F \) is a potentially nonlinear function available as a blackbox, \( x, y \) are vectors of discrete and continuous variables, \( \phi \) is a linear function, and \( P \) is a linear constraint. \( P \) can also be a conjunction of several constraints. Note that \( \land \) denotes logical conjunction (AND). This is a straightforward reformulation of the earlier inverse problem definition, with \( y \) being an explicit handle on the output. This reformulation allows a natural implementation using the constituent technologies. As shown in Section V-H, CNMA handles nonlinear objective functions and constraints by moving their nonlinearities inside the definition of \( F \).

A. CNMA Innovations

CNMA’s innovation is connecting the modeling power of NNs and constraint-solving power of MILP solvers into a learning-from-failure feedback loop in such a way that they do much of the work for us, permitting straightforward, efficient implementations of the following desirable features into a single, cohesive system.

1) Efficient Construction of a Surrogate Function: The complexity of NN is linear in the number of samples.

2) Efficient Constraint Solving Without Penalty Functions: This feature is enabled by the transformation of NNs with the ReLU activation function into an equivalent MILP. In addition, constraints are directly expressed in the MILP language and then efficiently solved by industrial-strength MILP solvers, such as CPLEX and GUROBI [11], [12].

3) Sample Efficiency: This feature is enabled by CNMA’s learning-from-failure feedback loop. A surrogate is learnt as an NN that is then transformed into an MILP. Using this MILP, a constrained optimization problem is solved and if the solution is unacceptable, it is used to find a new point to sample.

4) Optimization With Discrete and Continuous Variables: Function inputs, outputs, and constraints can all contain discrete and continuous variables. This feature is enabled by the use of both NNs and MILP solvers.

5) Solving Constraints Whose Evaluation Itself Requires Blackbox Evaluations: CNMA handles this by introducing new variables for components of constraints that must be evaluated via blackbox methods and shifts the blackbox estimation to the underlying surrogate that is created. See Section V-H.

6) Resilience to the Failure of Blackboxes to Compute Outputs: CNMA leverages the ability of NNs to learn despite missing information. It makes no assumptions about function continuity or smoothness. Fig. 2 shows...
an example of a function CNMA can optimize over.

7) **Parallelism:** A simple parallel version of CNMA, also presented in this article, improves the efficiency and quality of solutions over the sequential version, and also tries to steer it away from local optima. No restriction is placed on functions that can be optimized in parallel.

CNMA samples points in the domains of the function, and learns a NN surrogate of the function. By the Universal Approximation Theorem [33], NNs can approximate any continuous function, although in practice, they are also used to approximate noncontinuous, nonsmooth functions.

CNMA transforms that surrogate into an equivalent MILP [34] and constructs its conjunction with \( P(x, y) \). It optimizes \( \phi(x, y) \) subject to this conjunction using industrial-strength MILP solvers. It then checks the solution for correctness, i.e., whether it satisfies \( P(x, y) \) and whether the objective function is of acceptable value. If so, CNMA outputs the solution. If not, CNMA computes a new training instance from the solution and restarts. This “learning-from-failure” feedback loop has the effect of trying to sample the region of the domain relevant to solving the optimization problem. Thus, it reduces the number of function evaluations by orders of magnitude compared to that needed for learning the function over its entire domain.

A parallel version of CNMA uses multiple agents with each using a different NN architecture but operating off the same training set. Each independently computes the next best point to sample and adds it to the common training set. The resulting model diversity decreases the chances of getting stuck in local optima. Parallel NN training, MILP solving, and sample evaluation also contribute to improved performance over the sequential version.

Genetic algorithms can be combined with CNMA in a form of hybrid optimization to find solutions that may not be found by one or the other alone. This idea is thoroughly explored in a sister paper of the same title but Part II [28].

CNMA presents a novel method of addressing a major challenge posed in [49]: how to combine inductive and deductive reasoning in the design of cyber–physical systems. In CNMA, inductive reasoning is accomplished by NNs and deductive reasoning by MILP solvers, with the two tied together in a feedback loop.

This article is organized as follows. Section II discusses related work, in particular, highlighting the relationship of CNMA with BO/GP. Section III provides the necessary background. Section IV presents sequential and parallel CNMA and illustrates them with a simple example. Section V evaluates CNMA performance for seven nonlinear problems of 8 (two problems), 10, 15, 36 and 60 real-valued dimensions and one with 186 binary dimension. Its performance is compared with that of the skopt [35] implementation of BO/GP (abbreviated BO-S) and the scipy.optimize [36] implementation of NM (abbreviated NM-S), and Random Search. BO-S and NM-S are stable, off-the-shelf tools. Note, however, that BO-S did not return a solution for two problems and NM-S did not return one for three. Section VI concludes this article.

II. RELATED WORK

Surrogate-based methods are most closely related to CNMA. Of these methods, perhaps the most well-researched is BO/GP [9]. Let the objective function to be maximized be \( F(x) \), and let a blackbox be available to evaluate \( F(x) \). BO/GP is initialized with a covariance function \( k(x, y) \), also called a kernel. BO/GP is also initialized with a set \( S \) of samples in the domain of \( F \) and their associated values. With this information, BO/GP iteratively updates the posterior distribution of the underlying Gaussian Process, parameterized by \( \mu(x) \) and \( \sigma(x) \). Intuitively, \( \mu(x) \) is the mean of the values at \( x \), of all possible functions whose value for any sample \( v \) in \( S \) is \( F(v) \). \( \sigma(x) \) is the standard deviation of all these values. These two functions are combined in different ways to create a merit or acquisition function. This function is maximized using an optimization engine such as L-BFGS [37]. The value of \( x \) in the solution represents a new point to sample, relevant to the optimization problem that balances exploration with exploitation. It is added to \( S \) and the step repeats till a sample satisfying some halting condition is found. One such acquisition function is the Upper Confidence Bound UCB \( (x) = \mu(x) + \beta \ast \sigma(x) \) with \( \beta \geq 0 \). The construction of \( \mu(x) \) and \( \sigma(x) \) requires the inversion of the covariance matrix that lists \( k(u, v) \) for each pair \( (u, v) \) where \( u \) and \( v \) are samples in \( S \). In earlier versions of BO/GP, a matrix inversion method cubic in the number of samples was used [1], although faster methods on GPUs are reported in [15].

If \( F(x) \) is to be optimized subject to a constraint \( P(x, F(x)) \), \( P(x, F(x)) \) could be modeled as a penalty/barrier function and added to \( F(x) \). If \( P(x, F(x)) \) itself requires a blackbox evaluation, extensions of BO/GP have been proposed [16]–[19] that, in their inner loop, find \( x \) for which the likelihood of \( F(x) \) being the maximum and that of \( P(x, F(x)) \) being true is high.

Parallel versions of BO/GP have been proposed in [20]–[22]. Some, such as [20], assume function “additivity,” i.e., the function can be decomposed into a sum of functions on disjoint subsets of the function domain. Snoek et al. [38] reported the use of an NN as a surrogate but the surrogate remains inside the Gaussian Processes framework. It is not solved with an MILP solver. Daxberger et al. [39] reported a scheme for mixed discrete-continuous variables in BO/GP. To handle failure of function evaluation, Lee et al. [40] reports a scheme for learning problematic areas of the search space and avoiding those.

Like BO/GP, CNMA also builds a surrogate of \( F(x) \), say \( F_{nn}(x) \), from sampling the blackbox. The selection of the NN architecture is analogous to selection of the kernel and \( F_{nn}(x) \) is analogous to \( \mu(x) \). The MILP solver is analogous to an engine such as L-BFGS. It is directly used to find \( x \) that maximizes \( y \) such that \( F_{nn}(x) = y \land P(x, y) \). Any solution is used to compute the next point to sample.

Connecting NNs and MILP solvers in a learning-from-failure feedback loop permits efficient, straightforward implementation of above features: efficient surrogate function construction, sample efficiency, constraint solving without penalty functions, solving blackbox constraints, optimization with discrete and continuous variables, resilience to nonterminating function evaluations, and parallelism.

CNMA does not compute the variance of the surrogate it learns. Effectively, its merit function is \( \text{UCB}(x) \) with \( \beta = 0 \). There is, thus, a risk that it could get stuck in local optima finding more and more points around the current optima. CNMA provides two methods for trying to avoid this problem and searching globally. The first is to use a constraint stating that the objective function is above a threshold. Then, the MILP solver will not produce solutions for which the objective is below the threshold. The second is to introduce model diversity to reduce the chances of different models computing the same local optima. Model diversity is a byproduct of parallel CNMA whose multiple agents create their own models. Independently, parallel CNMA also contributes to
improved performance via parallel NN training, MILP solving and sample evaluation.

If the current surrogate is not good enough, then \( F_n(x) \) subject to \( P(x) \) may have no solution. In that case, CNMA generates a random point and restarts. How to mitigate such randomness is one problem of current research. Other future research problems include introducing additional diversity, e.g., by the use of bootstrapping, multifunction CNMA, finding an appropriate initial NN architecture and adapting that architecture as new samples are created, e.g., via the use of network compression [41].

We now discuss CNMA in detail.

III. BACKGROUND

A mixed-integer linear constraint is of the form \( a_0 \times x_0 + \cdots + a_k \times x_k \leq b \) where \( a_i \) and \( b \) are real-valued constants and the \( x_i \) are real-valued or integer-valued. An MILP is a set of such constraints with a linear objective function \( \phi(v_1, \ldots, v_m) \) where each \( v_i \) is a variable appearing in a constraint. An MILP solver finds values of all variables in the program optimizing the function while satisfying all constraints. It makes no distinction between input and output variables.

A NN is a set of layers with each layer consisting of a set of neurons. In the fully connected NN used in CNMA, each neuron in a layer is connected to each neuron in the previous layer. Such a network has one input layer, one output layer and zero or more hidden layers. When the values of neurons in the input layer are initialized, they are propagated forward to compute values of all neurons. The output layer can have multiple neurons allowing modeling of multioutput functions. Associated with the edge between two neurons is a weight. Associated with each neuron is a bias or intercept. The value of a hidden-layer neuron is a linear combination of its bias, values in the previous layer and weights of connecting edges, but passed through an activation function.

Activation functions give NNs the power to model nonlinear functions. We use the ReLU activation function \( \max(x, 0) \) because it can be converted into an MILP constraint using the big-M method [34]. By also modeling the overall system requirement as another mixed-integer linear constraint, scalable MILP solvers can be used to efficiently solve the NN along with the requirement. To allow NNs to model negative outputs, no activation function is applied at the output layer.

We now illustrate the above plan with a short example. To model the equation \( y = \max(x, 0) \) as an MILP, select a large number \( M \) and let an integer \( d \in \{0, 1\} \). Then, \( y = \max(x, 0) \) is equivalent to \( (y \geq 0 \land y \geq x \land y \leq x + M \land y \leq M(1 - d)) \).

Proof of Correctness of Transformation of ReLU Into MILP: To see how the MILP \( (y \geq 0 \land y \geq x \land y \leq x + M \land y \leq M(1 - d)) \) is equivalent to \( y = \max(x, 0) \), consider two cases. In the first case, let \( d = 0 \). The MILP simplifies to \( (y \geq 0 \land y \geq x \land y \leq x + M \land y \leq M(1 - d)) \) which can also be written as \( y = x \land x \geq 0 \). In the second case, let \( d = 1 \). The MILP simplifies to \( (y \geq 0 \land y \geq x \land y \leq x + M \land y \leq M(1 - d)) \) which can also be written as \( y = x \land x \geq 0 \). By combining these two cases, the MILP is equivalent to \( y = x \) if \( x \geq 0 \); otherwise, \( y = 0 \).

Using the above approach as a short example, to model the equation \( y = \max(x, 0) \) as an MILP, select a large number \( M \) and let an integer \( d \in \{0, 1\} \). Then, \( y = \max(x, 0) \) is equivalent to \( (y \geq 0 \land y \geq x \land y \leq x + M \land y \leq M(1 - d)) \).

A central innovation of CNMA is that the NN does not have to model \( F \) exactly to solve the optimization problem. To model it exactly would require an astronomical number of samples from \( F \)'s domain. Instead, CNMA tries to choose samples that are relevant to solving the optimization problem and are thus a tiny fraction of the number required for accurate modeling. Let \( M \) be a large enough number to cover the range of \( x \) and \( y \) where \( F \) is defined. Let \( \phi(x, y) \) subject to \( F(x, y) = y \land P(x, y) \). We can use an MILP solver to optimize \( \phi(x, y) \) subject to \( \text{nn_milp} \land P(x, y) \).

IV. METHODOLOGY

CNMA solves the problem of finding \( x \) and \( y \) that optimize \( \phi(x, y) \) such that \( F(x) = y \land P(x, y) \) where \( F \) is a nonlinear
function, \( x \) and \( y \) are vectors of discrete and continuous variables, \( \phi \) is a linear function, and \( P \) is a linear constraint. \( F \) is called the forward function. As shown in Fig. 4, CNMA uses a Sample Generator to sample points in \( F \)'s domain, evaluates those points, and creates a training set. These points are inputs to the NN ReLU Regression Engine that outputs a neural network \( nn \). This is transformed into an equivalent MILP \( \text{milp} \) by the NN-MILP transformer. \( \text{milp} \) is a surrogate or model of \( F \) based on current \( \text{samples} \). An MILP solver solves \( \phi(x, y) \) such that \( \text{milp} \land P(x, y) \) is true. If a solution is not found, the Sample Generator is called upon to extend \( \text{samples} \) with a new one in the hope of improving upon the current surrogate, and CNMA restarts. Otherwise, let \( (x^*, \hat{y}) \) be a solution. It is then checked for correctness, i.e., whether \( P(x^*, F(x^*)) \) holds. If it does, then the value of the objective function \( \phi(x^*, F(x^*)) \) is checked for acceptability, e.g., whether it is above or below the desired threshold or whether the evaluation budget has been reached. If the solution is acceptable, \( (x^*, F(x^*)) \) is output and CNMA halts. Otherwise, \( (x^*, F(x^*)) \) is added to \( \text{samples} \). If \( P(x^*, F(x^*)) \) is false, then \( (x^*, F(x^*)) \) is also added to \( \text{samples} \). Now, CNMA restarts. Note that CNMA can be used in pure constraint satisfaction mode by letting \( \phi(x, y) = 0 \) and in pure optimization mode by letting \( P(x, y) = true \).

The addition of \( (x^*, F(x^*)) \) if \( P(x^*, F(x^*)) \) does not hold is a form of learning from the failure to produce a surrogate of \( F \) that intersects \( P \). It has the effect of trying to restrict the sampling to only the part of \( F \)'s domain that is relevant to the satisfaction of \( P \). Thus, the sampling of \( F \) is reduced by many orders of magnitude over the fine-grained sampling needed to learn \( F \) over its entire domain. Even if \( P(x^*, F(x^*)) \) does hold, adding it to \( \text{samples} \) in the hopes of improving upon \( \phi(x) \) also restricts the sampling.

For clarity, the flowchart in Fig. 4 (and Algorithms 1 and 2 and Fig. 5) omit an important case: the failure to evaluate \( F \) for a given input either in the creation of the initial samples or in the evaluation of \( P(x^*, F(x^*)) \). In this case, CNMA just calls upon the Sample Generator to generate a new sample and restarts. NNs are resilient to missing data.

Algorithm 1 precisely defines the above plan. It is possible that CNMA could get stuck in local optima. There are two methods of steering it away from them. The first is adding a constraint on an upper or lower bound of the objective function so that the MILP solver finds solutions satisfying that constraint. The second is to use multiple concurrent copies of CNMA, each producing a different surrogate and, therefore, adding different “good” points to \( \text{samples} \). By pooling together these good points, the chances of finding the true optima can be improved. This method is a byproduct of Parallel CNMA that parallelizes sampling, training,
Algorithm 2 Parallel CNMA, Single Forward Function

Input: a problem definition of the form $\max_{x,y} \phi(x,y)$ s.t. $y = F(x) \wedge P(x,y)$, a method to randomly sample candidate solutions $x \in X$, maximum number $N$ of CNMA iterations, $M$ parallel solvers, $\phi$, $P$ are linear, $x,y$ are vectors of discrete and continuous variables, and $F$ is a potentially nonlinear function available as a blackbox.

Output: a solution, $(x^*, y^*)$, to the above problem

function CNMA($\phi$, $F$, $P$, $X$)
  samples ← $\{(x_i, F(x_i))\}_{i=1,2,\ldots,N}$, where $x_i$ denotes a random sample drawn from $X$
  solutions ← empty list
  for $i = 1, 2, \ldots, N$
    candidate_solutions ← empty list
    parallel for $j = 1, 2, \ldots, M$
      $nn$ ← a fully-connected ReLU regression network (with any arbitrary architecture), which takes in as input a vector $x \in X$ and attempts to predict $F(x)$; use samples to train this NN.
      $milp$ ← the MILP:
        $\max \phi(x,y)$ s.t. $nn_to_milp(nn) \wedge P(x,y)$
      $(x^*, y^*)$ ← solution to milp, obtained, e.g., via an MILP solver
        if $(x^*, y^*)$ is feasible
          candidate_solutions[$j$] ← $(x^*, y^*)$
        end
      evals ← in parallel, compute $(x_i, F(x_i))$ for each $x_i \in$ candidate_solutions
      $ninvalid$ ← 0
      for $j = 1, 2, \ldots, \text{size of (evals)}$
        $(x^*, y^*)$ ← candidate_solutions[$j$]
        if $(x^*, y^*)$ is infeasible
          $ninvalid$ ← $ninvalid + 1$
        else:
          Append $(x^*, y^*)$ to solutions
          if $P(x^*, y^*)$:
            Append $(x^*, y^*)$ to solutions
        end
      In parallel, append $ninvalid$ randomly drawn sample(s) $(x_i, F(x_i))$ to solutions
    end
  end
  return the best solution from solutions, sorting by $\phi(x,y)$
end

and solving to improve the quality and performance of the sequential version.

A. Parallel CNMA

As shown in Fig. 5, the CNMA algorithm can also be parallelized by having multiple instances of CNMA run simultaneously while periodically sharing information with each other. During each iteration, each CNMA solver uses the existing evaluated samples as training data to create a surrogate model of the forward function, $F$. Each solver may use a different NN architecture or initialize the NN weights differently, causing each solver to end up with a unique MILP problem to solve and produce a different point to sample next. Because there are many different NNs that all may fit the training data, running CNMA in parallel allows us to explore more of these diverse surrogate models in a single iteration. At the end of each iteration, all evaluated samples are shared among the solvers. See Algorithm 2 for a precise definition of Parallel CNMA.

B. Illustrating CNMA

The Rastrigin function is a common benchmark problem for optimization methods because it is highly nonlinear and has many local optima. We illustrate three different ways in which CNMA can be used to solve the optimization problem

$$\max_{x \in [-5,12.5,12]} F(x) = 10 + x^2 - 10 \cos(2 \pi x)$$

The true maximum of $F(x)$ is 40.353 at $x = -4.522$ and $x = 4.522$.

![Fig. 6. Maximizing Rastrigin.](image)

(a) During the first iteration the surrogate function predicts that the maximum is at $-5.12$. We evaluate Rastrigin at this point and the new sample, shown as a purple dot, is added to the training set.
(b) During iteration 2, the surrogate incorrectly predicts where the maximum value of the function is. The new sample, shown as a purple dot, is added to the training set.
(c) During iteration 3, the surrogate incorrectly predicts where the maximum value of the function is. The new sample, shown as a purple dot, is added to the training set.
(d) During iteration 4, the surrogate incorrectly predicts where the maximum value of the function is. The new sample, shown as a purple dot, is added to the training set.

| $x$        | $F(x)$  |
|------------|---------|
| $-3.495$   | 32.210  |
| $-2.436$   | 25.161  |

Fig. 6 shows the progression of CNMA at each iteration solving: maximize $\phi(x,y)$ s.t. $F(x) = y \wedge -5.12 \leq x \leq 5.12$. CNMA first generates two initial samples in the domain of $x$ to create the training set shown in Table 1.

CNMA then learns an NN from this set to create a surrogate of $F(x)$. Fig. 6 shows the surrogate function plotted in orange. CNMA converts this NN into an MILP and uses an MILP solver to solve it along with $P(x,y) = -5.12 \leq x \leq 5.12$ such that $y$ is maximized.

During the first iteration, the solution found is $x = -5.12$. This is where the maximum of the orange curve is within $x$'s domain. After checking the solution against the correct definition of $F(x)$, the dot shown in purple is added to the training set and a new NN is trained during Iteration 2. While CNMA is able to find a local maximum after just three iterations and five function calls, CNMA gets stuck here and a global maximum is not found even after 100 iterations.

To help CNMA explore outside this local maximum, we can use Parallel CNMA to add NN diversity. Fig. 7 shows how Parallel CNMA solves the same problem but is able to find the global maximum by using ten different CNMA workers simultaneously. During the first iteration, ten NNs, each initialized with different weights, are trained on the same two initial samples. This creates ten different MILP problems to be solved. Because each NN is slightly different, each MILP problem produces a different solution. The eighth NN predicts that the maximum value of the function is at $x = -4.571$, which is close to one of the true maxima at $x = -4.5229$. Since all samples are shared among all parallel workers, by

| $x$        | $F(x)$  |
|------------|---------|
| $-3.495$   | 32.210  |
| $-2.436$   | 25.161  |
problem is infeasible and a random sample is evaluated instead (depicted as a green dot). Many random samples continue to be generated until Iteration 37, where the surrogate function again overlaps with $P(x, y)$. After evaluating the MILP solution through the true function, this time $P(x, y)$ is satisfied. This causes the surrogate function to continue to overlap with $P(x, y)$ during subsequent iterations. The solution $x = 4.523$ with objective function $y = 40.353$ is found after 50 iterations and 52 function calls (2 initial + one each for 50 iterations).

V. EXPERIMENTAL RESULTS

For seven nonlinear design problems of 8 (two problems), 10, 15, 36, 60 real-valued dimensions and one with 186 binary dimension, we compare the performance of CNMA with the skopt [35] implementation of BO/GP (abbreviated BO-S), the scipy.optimize [36] implementation of NM (abbreviated NM-S), and with Random Search. We evaluated CNMA with one, five, and ten solvers in parallel. For NNs, we use the scikit-learn package [42]. We use a commercial MILP solver. All of these packages are stable-off-the-shelf.

For each problem, we allocate a fixed time budget that is, in our estimate, the longest a user would wait for a solution to that problem.

For each problem, we compare the optimization engines against two metrics. The first metric is the best value of the objective function computed within the problem’s time budget. It is evaluated by plotting the improvement of the objective function value against time, and comparing the best values from each engine.

The second metric is the minimum number of function evaluations needed to produce the best value of the objective function within the problem’s time budget. This metric is intended to capture the idea of “sample efficiency” since it can be quite expensive to evaluate blackboxes. This metric is evaluated by plotting the improvement of the objective function against the number of function evaluations, and comparing the minimum number of evaluations needed by each engine to produce the best value of the objective function.

We also compare Random Search by randomly generating the maximum number of samples used by any one of the optimization methods which found solutions.

For BO-S, we use the expected improvement (EI) acquisition function, and the Matern kernel. The following hyperparameters are automatically tuned by BO-S: 1) all kernel length scales; 2) covariance amplitudes; and 3) parameters of the independent identically distributed Gaussian noise added to the kernel.

For CNMA with one solver, we use an NN architecture with two hidden layers of 35 and ten neurons. For CNMA with five solvers, we use the same NN architecture in addition to ones with single hidden layers of 10, 30, 35, and 50 neurons. We use the same architectures (each repeated twice) for CNMA with ten solvers.

For BO-S and NM-S, we model constraints as a penalty function. Note that BO-S did not return a solution for two problems and NM-S did not return one for three. These problems are high-dimensional, have both discrete and continuous variables, and their blackboxes do not return outputs for some inputs. For other problems, CNMA improves the performance over BO-S, NM-S and Random Search by 1%–87%.

Some comparative visualizations are available at https://collab.peratonlabs.com/nonlinearbenchmarks/.
Fig. 9. Structure of wave energy propelled boat. Note the recursive relationship between the input and output variables of Boat and Hydrofoil.

A. Designing Wave-Energy-Propelled Boat

This example shows the ability of CNMA to optimize with functions that may fail to return a value for some inputs, as can happen with the xfoil [29] computational fluid dynamics simulator used here. Fig. 9 indicates how the rise and fall of a boat floating on a wave pulls and pushes at a hydrofoil below, causing a rotation about an axis. This rotation generates forward force during both the upward and downward wave motion; much like when swimmers flap flippers in a pool, their body is propelled forward. The marine robot is inspired by the Wave Glider [43]. The design problem is to compute the dimensions of the boat and hydrofoil which will maximize the steady-state forward sailing speed for a given wave condition. The equilibrium constraints are that the force generated by the hydrofoil equals that applied to the boat and that the glider and boat velocities are equal. An additional constraint is that the magnitude of the horizontal velocity be higher than that of the vertical velocity. Note the recursive relationship between the variables: force is an output of Hydrofoil but an input to Boat whereas velocities are outputs of the latter and inputs to the former.

The boat is modeled with two functions. The first is $\text{Hydrofoil}(\text{ch}, V_x, V_y, \theta) = f_{\text{Glider}}$, that computes the forward force output by a hydrofoil of length $\text{ch}$, moving through water at velocity $V_x$, $V_y$ at an angle of attack $\theta$. This is the force it applies to the boat. It is implemented with the computational fluid dynamics package xfoil. The second function is $(\text{length}, \text{beam}, \text{draft}, \text{ampl}, \text{period}, f_{\text{Boat}}) = [V_{\text{Boat}}, V_{\text{Boat}}]$. It outputs the steady-state forward speed of a boat given its 3-D dimensions: length, beam, and draft, the amplitude $\text{ampl}$ and period $\text{period}$ of the wave, and the forward force $f_{\text{Boat}}$, applied to it by the hydrofoil. This function is computed by a program encoding a solution to a differential equation. The first two equilibrium constraints are enforced by eliminating $V_x$ and $V_y$ and consolidating the two functions into the CNMA forward function $F(x) = y$ where

$$x = [\text{length}, \text{beam}, \text{draft}, \text{ampl}, \text{period}, f_{\text{Boat}}, \text{ch}, \theta]$$

$$y = [V_{\text{Boat}}, V_{\text{Boat}}, f_{\text{Glider}}].$$

$F$ calls Boat to compute $V_{\text{Boat}}$ and $V_{\text{Boat}}$ and then inputs them to Hydrofoil to compute $f_{\text{Glider}}$. The third equilibrium constraint is now $f_{\text{Boat}} = f_{\text{Glider}}$. To tolerate small force differences, the equality is modeled as $|f_{\text{Boat}} - f_{\text{Glider}}| \leq \epsilon * f_{\text{Boat}}$ where $\epsilon$ is set to 5%. Note that a constraint with an absolute value can be modeled as a pair of linear constraints [44]. Finally, $P(x, y) = (|f_{\text{Boat}} - f_{\text{Glider}}| \leq \epsilon * f_{\text{Boat}} \land V_{\text{Boat}} \geq V_{\text{Boat}})$ and $\phi(X, Y) = V_{\text{Boat}}$. In the optimization problem, $\text{ampl}$ and period are set to 4 and 7, respectively.

With 30 initial samples and a budget of 2500 s, with one, five, and ten solvers, CNMA is able to find a solution with a speed of 3.9036, 3.8993, and 3.8999 m/s, respectively. BO-S and NM-S are unable to find any feasible solutions given the same time budget. Out of 3890 randomly samples, none of them are valid solutions. Figs. 10 and 11 show the performance of CNMA with respect to time and number of function evaluations.

B. OpenAI Gym’s LunarLander: Robot System-Controller Codesign

Robots and their controllers are often designed separately. This can lead to design inconsistency whose resolution can be time consuming. If robots can be efficiently reconfigured and their controllers efficiently designed, we raise the possibility of system-controller codesign by solving a single optimization problem that encodes system and controller constraints and objectives. The lander we use, as depicted in Fig. 12, is part of the robotic benchmarks at OpenAI/Gym [45]. We assume
that the controller is proportional-integral-derivative (PID)-based. Hence, the controller design problem reduces to finding optimal values for the PID coefficients. From a mothership, the module is ejected with a certain force and then its engines fire both vertically and horizontally to guide it toward landing on the flat pad between two flagpoles. Our goal is to compute the system and controller design and initial position and force that would maximize the reward while satisfying constraints on a successful landing, time to land, and fuel usage.

The input vector to the CNMA forward function $F$ is $x = [\text{mep}, \text{sep}, \text{la}, \text{ld}, \text{lw}, \text{lh}, \text{lst}, \text{kp}_\text{alt}, \text{kd}_\text{alt}, \text{kp}_\text{ang}, \text{kd}_\text{ang}, \text{initial}_x, \text{initial}_y, \text{initial}_x, \text{initial}_y]$. The variables $\text{mep}$, $\text{sep}$, $\text{la}$, $\text{ld}$, $\text{lw}$, $\text{lh}$, and $\text{lst}$ are system design parameters denoting, respectively, main engine power, side engine power, leg away length, leg down length, leg width, leg height, and leg spring torque. The variables $\text{kp}_\text{alt}$, $\text{kd}_\text{alt}$, $\text{kp}_\text{ang}$, and $\text{kd}_\text{ang}$ are controller design parameters denoting, respectively, the $P$ and $D$ values for the vertical and horizontal engine controllers. The $I$ coefficient is set to 0. The variables $\text{initial}_x$, $\text{initial}_y$, $\text{initial}_x$, and $\text{initial}_y$ are initial condition parameters, denoting, respectively, the initial position and force at the time of lunar lander ejection from the mother ship. The forward function $F(x) = y$ simulates the trajectory of the lander defined by its system design parameters, PID parameters, and the initial conditions. It runs for a fixed number of time steps or until the lander lands or crashes. At each time step, it measures the "error," i.e., the distance between its current position and the landing point, and using the PID values computes engine firing actions to guide the lander toward the landing point. It outputs $y = [\text{fuel}, \text{time}, \text{success}, \text{reward}]$ denoting, respectively, the fuel used, time taken to land, whether the landing is safe, and the reward, a measure of the quality of the landing, as defined by OpenAI/Gym. The codesign problem is now to maximize the objective function $\phi(x, y) = \text{reward}$ subject to $F(x) = Y \land P(x, y)$ where $P(x, y) = (\text{success} = 1 \land \text{fuel} \leq 75 \land \text{time} \leq 10)$.

With 100 initial samples and a time budget of 1500 s, CNMA finds solutions with rewards of 437.139, 469.589, and 465.136 with one, five, and ten solvers, respectively. In the same amount of time, BO-S only finds a solution with a reward of 395.118 and NM-S only finds a solution with a reward of 409.857. Out of 1966 random samples, the best solution which meets the constraints has a reward of 446.199. Figs. 14 and 15 show the performance of CNMA, BO-S, NM-S, and Random Search with respect to time and number of function evaluations.

C. Optimizing Hexapod Gaits

We now address a problem inspired by the work of Cully et al. [46] for adapting robot gait to failures in the field. The robot is six-legged with each leg consisting of three segments. Associated with each leg $i$ is a vector of six parameters $(a_1, a_2, \phi_1, \phi_2, \tau_1, \tau_2)$ with each $\alpha, \phi, \tau \in [0, 1]$. These parameters determine, respectively, the amplitude phase and duty cycle of the walking signal sent to the first two legs every 30 ms. The walking signal for the third segment is the inverse of that for the second so does not need independent control parameters. The hexapod (as shown in Fig. 13) controller is defined by the six parameters for each leg for a total of 36 parameters, and fully determines the hexapod gait. Using the hexapod simulator in [46], we define a CNMA forward function $\text{hexapod}(x) = y$ that takes a controller $x = [c_0, \ldots, c_{35}]$ as input, simulates the hexapod gait for 5 s and outputs a vector $y = [\text{speed}_x, b_1, b_2, b_3, b_4, b_5, b_6]$ where $\text{speed}_x$ is the hexapod’s $x$-axis displacement in meters divided by 5.0 and each $b_i$ is the fraction of the time leg $i$ is in contact with the ground. Hexapod speeds above 0.20 m/s are hard to find with Random Search [46].
If a hexapod leg is broken then we would like to find a new controller that can achieve a speed of above 0.20 m/s while satisfying any new constraints on the movement. Let us assume leg 1 is broken. Then, we might constrain its contact with the ground to be the least of that of all the legs. The problem is now: maximize speed subject to hexapod([l_0, \ldots, l_{35}]) = \left[\text{speed, } b_1, \ldots, b_6\right] \wedge b_1 \leq b_2 \wedge b_1 \leq b_3 \wedge b_1 \leq b_4 \wedge b_1 \leq b_5 \wedge b_1 = b_6 \text{ and stop when the speed is close enough to 0.20 m/s.} As can be seen above, the baseline controller does not satisfy the constraint.

We can try searching over [0, 1)^{36} for a new controller. However, another option is to search just in the neighborhood of the baseline controller. We change the bounds for each field in the baseline controller to be within 0.1 of its current value, subject to the lower bound being at least 0 and the upper bound at most 1.

With two initial samples and a time budget of 10000 s, CNMA finds solutions with speeds of 0.2043, 0.2579, and 0.2619 m/s with one, five, and ten solvers, respectively. BO-S finds a solution with a speed of 0.2064 m/s while NM-S is not able to find any valid solutions. Out of 4422 randomly generated samples, the best solution has a speed of 0.2058 m/s. Figs. 16 and 17 show the performance of CNMA, BO-S, NM-S, and Random Search with respect to time and number of function evaluations.

D. Acrobot Design

The Acrobot [45], as shown in Fig. 18, is a two-link robot arm with a single actuator placed at the elbow. Initially, the links hang downward. The Acrobot’s goal is to execute a series of actions that vertically orients and balances both links. The Acrobot problem is well studied, and is known to be challenging to solve. In Fig. 18, system design variables m_i, l_i, and I_i; denote, respectively, the mass, moment of inertia, and center of mass location of link i. The CNMA function F takes as input a system design vector x = [m_1, m_2, I_1, I_2, l_{c_1}, l_{c_2}, l_1, l_2], runs iterative linear-quadratic regulator (LQR) as our controller on an Acrobot system with this vector and returns t_{stabilize}, the total time taken to balance the system. t_{stabilize} is listed as t_{stabilize} in the chart below. The problem is to find a system design that minimizes \phi(x, y) = t_{stabilize} subject to F(x) = Y \land P(x, y) where

\[ P(x, y) = (l_1 = l_2 \land 0.1 \leq m_1, m_2 \leq 3.0 \land 0.1 \leq l_1, l_2 \leq 3.0 \land 0.3 \leq l_{c_1}, l_{c_2} \leq 0.7). \]

The constraint l_1 = l_2 reflects OpenAI/Gym’s implementation of the Acrobot problem.

With 20 initial samples and a budget of 5000 s, CNMA finds solutions with objective function values 3.4, 3.2, and 2.8 with one, five, and ten solvers, respectively. In the same amount of time, BO-S is able to find a solution with an objective function of 3.2 and NM-S is able to find a solution with an objective function of 8.6. Out of 500 randomly generated solutions, the best solution found has an objective function value of 4.2. Figs. 19 and 20 show the performance of CNMA, BO-S, NM-S, and Random Search with respect to time and number of function evaluations.

E. Optimal Sensor Placement for Power Grids

We now design a system in which all inputs are discrete but the output is continuous. Placing current and voltage sensors on a power grid can help identify which power lines are down during power outages. However, placing these sensors can be expensive and their number has to be limited. Given a limited sensor budget, the problem is to determine where to place the sensors such that their readings give the best chance of predicting the line failure pattern. We define a forward function F that takes in a sensor placement x of bits and outputs y = [ambiguity], the “ambiguity” of the placement. In x, if a sensor is placed at line i, then x[i] = 1; else, x[i] = 0. For a given x, we simulate all single-line failures and record the associated set of readings at the sensors placed in x. If a reading set appears more than once then it is ambiguous since more than one failure can cause it. We then divide the number of ambiguous reading sets by the total number of reading sets.

Fig. 16. Speed versus time for Hexapod. NM-S stops after 207 s due to early convergence. NM-S is not shown because it is not able to find any valid solutions. After 2000 s CNMA with 5 and 10 solvers outperforms BO-S.

Fig. 17. Speed versus number of function evaluations for Hexapod. NM-S is not shown because it is not able to find valid solutions. CNMA outperforms Random Search for any given budget of function evaluations less than 4422. CNMA with 5 and 10 solvers matches or outperforms BO-S given a budget of function evaluations between 400 and 600.

Fig. 18. Acrobot schematic.
Fig. 19. Stabilization time versus time for Acrobot. CNMA outperforms NM-S when given a time budget of less than 5000 s. CNMA matches or outperforms BO-S when given a time budget of less than 5000 s.

Fig. 20. Stabilization time versus number of function evaluations for Acrobot. CNMA matches or outperforms NM-S, BO-S, and Random Search when given the same budget of function evaluations.

to compute [ambiguity] of \( x \). Sensor readings are computed by the power line simulator OpenDSS [47], which takes in a model of the power grid, sensor placement, and a line failure pattern and outputs sensor readings. The objective function to minimize is \( \phi(x, y) = \text{ambiguity} \) subject to the constraint that the number of sensors placed at most meets some budget. For the Bus118 power grid with 186 power lines, and a budget of 50 sensors, the constraint is specified as \( P(x, y) = \sum x \leq 50 \).

With 30 initial samples and a time budget of 1500 s, CNMA finds solutions with objective function values 0.04813, 0.02139, and 0.02139 with one, five, and ten solvers, respectively. In the same amount of time, BO-S and NM-S are not able to find solutions that meet the constraints. Since these two only work with real-valued variables, we round their outputs to 1 or 0 to evaluate the forward function. Out of 460 randomly generated samples, none of them are valid solutions. Figs. 21 and 22 show the performance of CNMA, BO-S, NM-S and Random Search with respect to time and number of function evaluations.

F. Rover Path Planning

This problem, defined in [20], involves finding a trajectory, such as the one shown in Fig. 23, for a robot with starting and end goal positions. The trajectory is specified by a set of 30 2-D points that are fit by a BSpline to define a path. Each trajectory also has an associated cost which penalizes obstacle collisions and should be minimized. The CNMA function \( F \) takes as input the 2-D coordinates of the 30 points (60 inputs) that define a path and outputs the cost of the trajectory. Each input ranges from 0.0 to 1.0. The objective is to maximize the negative cost of the trajectory.

With two initial samples and a time budget of 9000 s, CNMA finds solutions with costs of 1.148, 0.997, and 0.778 with one, five, and ten solvers, respectively. Given the same time budget, BO-S finds a solution with a cost of 3.530
and NM-S finds a solution with a cost of 5.867. Out of 4452 random samples, the best solution has a cost of 0.788. Figs. 24 and 25 show the performance of CNMA, BO-S, NM-S, and Random Search with respect to time and number of function evaluations.

### G. Polak3

This optimization benchmark [48] involves minimizing the maximum value of ten different transcendental functions. The CNMA forward function \( F(x) = y \) takes in ten values, \( x_1, \ldots, x_{10} \), each \( x_i \in [-1, 1] \), and outputs \( y \), the maximum value of the ten transcendental functions. The best known minimum is 5.93. An example of one of the transcendental functions is

\[
\begin{align*}
&\quad +0.2 * \left(e^{(x_5 - np \sin(0.0+5.0+5.0))} + (x_5 - \sin(0.0+5.0+5.0))\right) \\
&\quad +0.1666 * \left(e^{(x_6 - \sin(0.0+6.0+6.0))} + (x_6 - \sin(0.0+6.0+6.0))\right) \\
&\quad +0.1428 * \left(e^{(x_7 - \sin(0.0+7.0+7.0))} + (x_7 - \sin(0.0+7.0+7.0))\right) \\
&\quad +0.125 * \left(e^{(x_8 - \sin(0.0+8.0+8.0))} + (x_8 - \sin(0.0+8.0+8.0))\right) \\
&\quad +0.1111 * \left(e^{(x_9 - \sin(0.0+9.0+9.0))} + (x_9 - \sin(0.0+9.0+9.0))\right) \\
&\quad +0.1 * \left(e^{(x_{10} - \sin(0.0+10.0+10.0))} + (x_{10} - \sin(0.0+10.0+10.0))\right) \\
&\quad +0.0909 * \left(e^{(x_{11} - \sin(0.0+11.0+11.0))} + (x_{11} - \sin(0.0+11.0+11.0))\right)
\end{align*}
\]

With 20 initial samples and a time budget of 2000 s, with one, five, and ten solvers, CNMA finds solutions with objective function values 5.97, 6.06, and 5.98, respectively. Given the same time budget, BO-S finds a solution with an objective function value of 6.08 and NM-S finds a solution with an objective function value of 6.99. Out of 2680 randomly generated samples, the best solution has an objective function of 6.81. Figs. 26 and 27 show the performance of CNMA, BO-S, NM-S, and Random Search.
Search with respect to time and number of function evaluations.

H. Modeling Nonlinear Constraints and Objective Functions

To model a nonlinear constraint $P$, we add, for each nonlinear expression in $P$, an extra output to $F$ denoting the value of the expression and then express $P$ as a linear expression in $P$, we would have added an extra output to $F$. Modeling Nonlinear Constraints and Objective Functions

\[ F(x_1, x_2) = (x_1^2 - 2x_2) + x_1 + 2x_2 + 1.5 \]

We define a function $F(x_1, x_2)$ that produces two outputs $v_1, v_2$ computing, respectively, $c_1(x_1, x_2)$ and $c_2(x_1, x_2)$. Then, with CNMA, we solve the problem of minimizing $x_1 + x_2$ subject to $F(x_1, x_2) = [v_1, v_2] \land v_1 \geq 0, v_2 \geq 0$. In 84 evaluations (10 initial + 74 additional), CNMA finds a solution of 0.6003. The minimum is 0.599. Our NN architecture has 35 and ten neurons in the two hidden layers. The important point to note is that we can solve this problem through specification, not by changing CNMA. If the objective function $g$ had been nonlinear, we would have added an extra output to $F$ and minimized that.

VI. CONCLUSION

System design tools are often only available as blackboxes with complex nonlinear relationships between inputs and outputs. This article presents CNMA, a new constrained optimization method for blackboxes for solving the inverse problem of finding designs from requirements on output.

CNMA’s innovation is connecting the modeling power of NNS and constraint-solving power of MILP solvers into a learning-from-failure feedback loop in such a way that they do much of the work for us, permitting straightforward implementations of several desirable features into a single, cohesive system: efficient surrogate function construction, sample efficiency, constraint solving without penalty functions, solving blackbox constraints, optimization with discrete and continuous variables, resilience to nonterminating function evaluations, and parallelism.

If a large and deep NN is needed to model a complex function, its MILP equivalent may not be efficiently solvable. However, CNMA does not need to model the function in its entire domain. It only needs to model it in the part of the domain relevant to solving the constrained optimization problem. If this region is not too complex, a smaller NN is adequate so that its MILP equivalent could be efficiently solvable. This region is automatically computed by CNMA. As we have seen, the largest network used had [10], [35] neurons in its hidden layers and most problems start with few tens of initial samples, some even with just two. In fact, a large or deep NN may be detrimental to performance as it would overfit the small number of points CNMA samples.

CNMA is evaluated for seven nonlinear design problems of 8 (two problems), 10, 15, 36, and 60 real-valued dimensions and one with 186 binary dimension. It is shown that CNMA improves upon stable, off-the-shelf implementations of BO/GP (BO-S), Nelder Mead (NM-S), and Random Search by 1%–87% for a fixed time and function evaluation budget. Note, however, that BO-S did not return a solution for two problems and NM-S did not return one for three. Future research problems include introducing additional diversity, e.g., via bootstrapping, multifunction CNMA, and finding a good initial NN architecture and adapting it as new samples are created.

ACKNOWLEDGMENT

The authors thank the anonymous reviewers and Antoine Cully, Robert Gramacy, Matthias Poloczek, Robert Vanderbei, John Chinnick, Nick Sahinidis, Jaime Fisac, Nicholas Kraus, Taylor Njaka, Jeremy Cohen, Todd Hunter, Mengdi Wang, and Chetan Narain, for helpful comments. The views, opinions, and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

REFERENCES

[1] C. Rasmussen and C. Williams, Gaussian Processes for Machine Learning. Cambridge, MA, USA: MIT Press, 2006.
[2] D. Eriksson and M. Poloczek, “Scalable constrained Bayesian optimization,” in Proc. Adv. Neural Inf. Process. Syst., vol. 31, pp. 393–401, 2018.
[3] Z. Wang, C. Gehring, P. Kohli, and S. Jegelka, “Batched large-scale Bayesian optimization in high-dimensional spaces,” in Proc. Int. Conf. Artif. Intell. Stat., 2019, pp. 730–738.
[4] V. Picheny, R. Gramacy, S. Wild, and S. Le Digabel, “Bayesian optimization in high-dimensional spaces,” in Proc. Adv. Neural Inf. Process. Syst., vol. 31, pp. 393–401, 2018.
[5] S. Singer and S. Singer, “Efficient implementation of the Nelder–Mead search algorithm,” Appl. Numer. Anal. Comput. Math., vol. 1, no. 2, pp. 524–534, 2004.
[6] R. Gramacy, Surrogates: Gaussian Process Modeling, Design, and Optimization for the Applied Sciences. Boca Raton, FL, USA: CRC Press, 2020.
K. Kandasamy, A. Krishnamurthy, J. Schneider, and B. Póczos, “Parallelised Bayesian optimisation via Thompson sampling,” in Proc. Int. Conf. Artif. Intell. Stat., 2018, pp. 133–142.

I. Pan, M. Babaei, A. Korre, and S. Durucan, “Artificial neural network based surrogate modelling for multi-objective optimisation of geological CO₂ storage operations,” Energy Procedia, vol. 63, Dec. 2014, Art. no. 3483–3491.

B. Grimstad and H. Andersson, “ReLU networks as surrogate models in mixed-integer linear programs,” 2019. arXiv:1907.03140.

S. P. Mohanty, “Metamodel-based fast AMS-SoC design methodologies,” in Nanoelectronic Mixed-Signal System Design. New York, NY, USA: McGraw-Hill, 2015.

O. Garitselov, S. P. Mohanty, and E. Kougianos, “Fast-accurate non-polynomial metamodeling for nano-CMOS PLL design optimization,” in Proc. 25th Int. Conf. VLSI Design, 2012, pp. 316–321.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” IEEE Trans. Evol. Comput., vol. 6, no. 2, pp. 182–197, Apr. 2002.

P. Terway, K. Hamidouche, and N. K. Jha, “Fast design space exploration of nonlinear systems: Part II,” 2021. arxiv:2104.02464.

Xfoil, Subsonic Airfoil Development System. Accessed: Mar. 25, 2021. [Online]. Available: https://bit.ly/3rzxFh5.

D. P. Bertsekas, “On penalty and multiplier methods for constrained minimization,” SIAM J. Control Optim., vol. 14, no. 2, pp. 216–235, Feb. 1976.

J. Nocedal and S. Wright, “Parallelised Bayesian optimisation via Thompson sampling,” in Proc. Int. Conf. Artif. Intell. Stat., 2018, pp. 133–142.

S. P. Mohanty, “Metamodel-based fast AMS-SoC design methodologies,” in Nanoelectronic Mixed-Signal System Design. New York, NY, USA: McGraw-Hill, 2015.

O. Garitselov, S. P. Mohanty, and E. Kougianos, “Fast-accurate non-polynomial metamodeling for nano-CMOS PLL design optimization,” in Proc. 25th Int. Conf. VLSI Design, 2012, pp. 316–321.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” IEEE Trans. Evol. Comput., vol. 6, no. 2, pp. 182–197, Apr. 2002.

P. Terway, K. Hamidouche, and N. K. Jha, “Fast design space exploration of nonlinear systems: Part II,” 2021. arxiv:2104.02464.

M. Akintunde, A. Lomuscio, L. Maganti, and E. Pirovano, “Reachability analysis for neural agent-environment systems,” in Proc. Int. Conf. Principles Knowl. Represent. Reason., 2018, pp. 184–193.

Scikit-Optimize. Accessed: Mar. 25, 2021. [Online]. Available: https://scikit-optimize.github.io/stable/.

Scipy. Optimize. Accessed: Mar. 25, 2021. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/optimize.html.

L. Guilhoto. An Overview of Artificial Neural Networks for Mathematicians. Accessed: Mar. 25, 2021. [Online]. Available: https://bit.ly/3ybBUU.

A. Doboli and A. Vemuri, “Behavioral modeling for high-level synthesis of analog and mixed-signal systems from VHDL-AMS,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 22, no. 11, pp. 1504–1520, Nov. 2003.

K. Kandasamy, A. Krishnamurthy, J. Schneider, and B. Póczos, “Parallelised Bayesian optimisation via Thompson sampling,” in Proc. Int. Conf. Artif. Intell. Stat., 2018, pp. 133–142.

I. Pan, M. Babaei, A. Korre, and S. Durucan, “Artificial neural network based surrogate modelling for multi-objective optimisation of geological CO₂ storage operations,” Energy Procedia, vol. 63, Dec. 2014, Art. no. 3483–3491.

B. Grimstad and H. Andersson, “ReLU networks as surrogate models in mixed-integer linear programs,” 2019. arXiv:1907.03140.

S. P. Mohanty, “Metamodel-based fast AMS-SoC design methodologies,” in Nanoelectronic Mixed-Signal System Design. New York, NY, USA: McGraw-Hill, 2015.

O. Garitselov, S. P. Mohanty, and E. Kougianos, “Fast-accurate non-polynomial metamodeling for nano-CMOS PLL design optimization,” in Proc. 25th Int. Conf. VLSI Design, 2012, pp. 316–321.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” IEEE Trans. Evol. Comput., vol. 6, no. 2, pp. 182–197, Apr. 2002.

P. Terway, K. Hamidouche, and N. K. Jha, “Fast design space exploration of nonlinear systems: Part II,” 2021. arxiv:2104.02464.

Xfoil, Subsonic Airfoil Development System. Accessed: Mar. 25, 2021. [Online]. Available: https://bit.ly/3rzxFh5.

D. P. Bertsekas, “On penalty and multiplier methods for constrained minimization,” SIAM J. Control Optim., vol. 14, no. 2, pp. 216–235, Feb. 1976.

J. Nocedal and S. Wright, “Parallelised Bayesian optimisation via Thompson sampling,” in Proc. Int. Conf. Artif. Intell. Stat., 2018, pp. 133–142.

M. Akintunde, A. Lomuscio, L. Maganti, and E. Pirovano, “Reachability analysis for neural agent-environment systems,” in Proc. Int. Conf. Principles Knowl. Represent. Reason., 2018, pp. 184–193.

Scikit-Optimize. Accessed: Mar. 25, 2021. [Online]. Available: https://scikit-optimize.github.io/stable/.

Scipy. Optimize. Accessed: Mar. 25, 2021. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/optimize.html.

L. Guilhoto. An Overview of Artificial Neural Networks for Mathematicians. Accessed: Mar. 25, 2021. [Online]. Available: https://bit.ly/3ybBUU.

A. Doboli and A. Vemuri, “Behavioral modeling for high-level synthesis of analog and mixed-signal systems from VHDL-AMS,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 22, no. 11, pp. 1504–1520, Nov. 2003.

K. Kandasamy, A. Krishnamurthy, J. Schneider, and B. Póczos, “Parallelised Bayesian optimisation via Thompson sampling,” in Proc. Int. Conf. Artif. Intell. Stat., 2018, pp. 133–142.

I. Pan, M. Babaei, A. Korre, and S. Durucan, “Artificial neural network based surrogate modelling for multi-objective optimisation of geological CO₂ storage operations,” Energy Procedia, vol. 63, Dec. 2014, Art. no. 3483–3491.

B. Grimstad and H. Andersson, “ReLU networks as surrogate models in mixed-integer linear programs,” 2019. arXiv:1907.03140.

S. P. Mohanty, “Metamodel-based fast AMS-SoC design methodologies,” in Nanoelectronic Mixed-Signal System Design. New York, NY, USA: McGraw-Hill, 2015.

O. Garitselov, S. P. Mohanty, and E. Kougianos, “Fast-accurate non-polynomial metamodeling for nano-CMOS PLL design optimization,” in Proc. 25th Int. Conf. VLSI Design, 2012, pp. 316–321.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” IEEE Trans. Evol. Comput., vol. 6, no. 2, pp. 182–197, Apr. 2002.

P. Terway, K. Hamidouche, and N. K. Jha, “Fast design space exploration of nonlinear systems: Part II,” 2021. arxiv:2104.02464.

Xfoil, Subsonic Airfoil Development System. Accessed: Mar. 25, 2021. [Online]. Available: https://bit.ly/3rzxFh5.

D. P. Bertsekas, “On penalty and multiplier methods for constrained minimization,” SIAM J. Control Optim., vol. 14, no. 2, pp. 216–235, Feb. 1976.

J. Nocedal and S. Wright, “Parallelised Bayesian optimisation via Thompson sampling,” in Proc. Int. Conf. Artif. Intell. Stat., 2018, pp. 133–142.