INFORM: Inverse Design Methodology for Constrained Multi-objective Optimization

Prerit Terway, and Niraj K. Jha, *Fellow, IEEE*

***Abstract*—Many system design methods use population-based optimization or a surrogate model for solving constrained multi-objective optimization. When designing a system with multiple objectives and constraints, the designer may first be interested in understanding the trade-offs among different objectives from a small number of simulations. In the next step, the designer may focus on specific regions of interest in the design space near a set of non-dominated solutions to further improve performance on the targeted objectives. This may help make the search process sample-efficient. We propose INFORM: a two-step approach for sample-efficient constrained multi-objective optimization of real-world nonlinear systems. In the first step, we modify a genetic algorithm (GA) to make the design process sample-efficient. We inject candidate solutions into the GA population using inverse design methods instead of determining the candidate solutions for the next generation using only crossover and mutation, as is done in standard GA. We present three types of inverse design techniques based on a (i) neural network verifier, (ii) neural network, and (iii) Gaussian mixture model. The candidate solutions for the next generation are thus a mix of those generated using crossover/mutation and solutions generated using inverse design. At the end of the first step, we obtain a set of non-dominated solutions. In the second step, we choose the regions of interest around the non-dominated solutions to further improve the objective function values using inverse design methods. We demonstrate the efficacy of INFORM through synthesis of nonlinear systems and analog circuits. The experimental results show that INFORM reduces synthesis time by up to 29× and improves the value of the objective function by up to 33% compared to a state-of-the-art baseline design methodology.**

***Index Terms*—Active learning; constrained multi-objective optimization; evolutionary algorithm; Gaussian mixture model; inverse design; mixed-integer linear program; neural networks; sample efficiency; system synthesis.**

I. Introduction

System design involves selecting system configuration(s) from a design space to solve a constrained multi-objective optimization (CMOO) problem. Using CMOO, we aim to obtain designs that achieve the best value of the objective functions (e.g., power, energy, fuel cost, etc.) while satisfying all constraints on system configuration (e.g., component values) and simulation outputs (e.g., noise, gain, peaking, etc.). Current design techniques often rely on evolutionary algorithms (EAs) to solve CMOO problems. EAs evaluate a diverse set of candidate solutions over multiple generations to avoid local minima. However, EAs are sample-inefficient and require numerous runs to find the best solution to the CMOO problem. Bayesian optimization (BO) is another widely used sample-efficient system design technique. It uses a surrogate model to actively learn the next candidate solution(s) for simulation. Although BO is sample-efficient, the surrogate model building complexity is $O(N^3)$, where N denotes the number of samples. Furthermore, most existing methods try to find the entire Pareto front [1] (which is typically unknown for a real-world system) rather than focus on specific regions near the set of non-dominated solutions that may be of most interest to the designer. A solution is non-dominated if no other design is better in at least one and better or equal in all other objectives with respect to the given solution.

We propose INFORM (Inverse Design Methodology for Constrained Multi-objective Optimization), a two-step sample-efficient synthesis methodology for solving CMOO problems. In Step 1, we modify a genetic algorithm (GA) to generate candidate solutions for the next generation using inverse design methodologies, in addition to solutions generated using crossover and mutation. Inverse design methodologies use a surrogate model to generate candidate solution(s) based on the desired system objectives and constraints. This contrasts with the typical use of a surrogate model in the literature. Typically, surrogate models evaluate multiple candidate solutions to select the most promising ones for simulation. Injection of candidate solutions using surrogate models exploits system response from all past simulations, rather than just using information from the previous generation of solutions, as done in a GA. We use three types of inverse design techniques: (i) neural network (NN) verifier, (ii) NN, and (iii) Gaussian mixture model (GMM), and their various combinations, resulting in a total of seven inverse design methods. The NN-Verifier (NN-Ver.) converts an NN surrogate model with ReLU activation to a mixed-integer linear program (MILP). NN-based inverse design uses a surrogate model to map the simulation outputs (e.g., system objectives/constraints) to the simulation inputs (e.g., system component values). GMM uses the joint probability distribution of simulation inputs and corresponding simulation outputs to generate candidate solutions. We dynamically determine the number of mixture components and the weight of each component using previous simulations. GMM-based candidate solutions correspond to the expected value of the conditional distribution of the component values, given the desired objectives/constraints. We use inverse designs to enhance system objectives and constraints with the aim of obtaining solutions that are better or are non-dominated with respect to solutions obtained through past simulations. We terminate Step 1 upon meeting some stopping criteria and obtaining a set of non-dominated solutions.

In Step 2, we harness the benefits of inverse designs to focus only on specific regions of interest around the set of non-dominated solutions obtained in Step 1. This enables targeted simulations, making the design process sample-efficient. We use the same inverse design methodologies that are used in Step 1 to generate solutions that dominate the solution(s) selected in Step 1. We aim to continuously obtain better solutions until we arrive at stopping criteria that terminate Step 2. The designer

This work was done under a contract from the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory (AFRL). 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. Prerit Terway and Niraj K. Jha are with the Department of Electrical and Computer Engineering, Princeton University, Princeton, NJ 08544, USA, e-mail:{pterway, jha}@princeton.edu.

can choose the number of candidate solutions that should be generated in each iteration of Step 2 to balance the time to generate solutions and the simulation cost.

We summarize the main contributions of INFORM next.

- Its Step 1 achieves a higher hypervolume compared to population-based optimization using the well-known GA algorithm: NSGA-II [2]. Hypervolume measures the size of the dominated space bounded by a reference point [3].
- Its Step 2 enables easy adaptation to different designer preferences by focusing on the specific regions of interest around the set of non-dominated solutions obtained in Step 1. Using Step 2 can also drive an existing solution towards another solution with different specifications, thus avoiding search from scratch.
- Use of inverse designs enables it to directly handle multiple objective/constraints on simulation inputs and outputs instead of using a pseudometric like entropy or expected hypervolume improvement (EHVI) [4]. Note that the computational complexity of obtaining the hypervolume increases with an increase in the number of objectives.
- It generates multiple candidates solutions (like NSGA-II), thus enabling the use of multiple cores for simulation.
- Its sample-efficient and lightweight design methodology does not require graphical processing units (GPUs), thus reduces optimization cost.

The rest of the article is organized as follows. In Section II, we discuss related work. Section III provides the necessary background, followed by a simple motivational example in Section IV. In Section V, we present the sample-efficient CMOO methodology. We evaluate the system synthesis methodology in Section VI. We discuss the highlights and limitations of INFORM in Section VII. Section VIII concludes the article.

II. Related Work

Next, we review prior work on solving CMOO problems. We cover weighted sum-based techniques, e.g., reinforcement learning (RL) and BO, that convert the CMOO problem into a weighted sum of objectives and constraints. We also discuss optimization methods like EA and recent versions of BO that return a Pareto front as the solution to CMOO.

A. Weighted-sum Optimization

RL uses the notion of reward to determine the appropriate action in each state using a policy to solve the optimization problem. Most RL methods require formulating the reward as a weighted sum of objectives and constraints. Wang et al. [5] use deep deterministic policy gradient (DDPG) with an encoder-decoder framework for the actor to synthesize analog circuits. AutoCkt [6] uses deep RL to learn the trade-offs among different objectives across the design space when designing electrical circuits. RL in conjunction with graph NN is used to reduce the synthesis time for chip floorplanning in [7]. Circuit-GNN [8] uses a graph NN to address forward and inverse design problems for circuit design across different topologies.

BO is another widely used technique for solving black-box optimization problems. It uses a surrogate model to learn the system response using past data. An acquisition function determines the candidate solution using the surrogate model. The acquisition function balances exploration of a new region with exploitation of past knowledge when generating the candidate solution. Lyu et al. [9] exploit disagreement between different acquisition functions to propose a batch of candidate solutions at each iteration to synthesize amplifiers.

B. Multi-objective Optimization (MOO)

MOO handles the objective(s) and constraint(s) separately. Solving CMOO results in solutions that represent the best trade-off among the outputs. The solutions satisfy the design constraints and optimize the output objectives. EA is a widely used technique that evolves a set of initial candidate solutions across generations to return the Pareto front, indicating the best trade-off among the objectives. NSGA-II [2], a prominent EA, uses fast non-dominated sorting to reduce the computational complexity by an order of magnitude. Deb et al. [10] propose a reference point-based many-objective EA called NSGA-III to place emphasis on non-dominated population members to solve CMOO problems.

Recent works modify the BO engine to solve MOO problems. Daulton et al. [4] present a parallel version of EHVI to derive multiple candidate solutions at each iteration to solve CMOO problems. Lyu et al. [11] use Gaussian process as the surrogate model for multiple objectives. The acquisition function is derived using the Pareto front of the lower confidence bound of the multiple objectives. A Bayesian NN is combined with BO to obtain candidate solutions based on a "pseudo" Pareto front in [12]. However, Refs. [11] and [12] do not handle constraints on the output. WATSON [13] combines multiobjective genetic optimization and multivariate regression to obtain the Pareto front in analog circuit design. ENSGA [14] uses variable population size in a GA and Monte Carlo simulations to assess the sensitivity of the solutions on the Pareto front to process variations in the synthesis of amplifier and oscillators. CNMA [15] and ASSENT [16] use inverse designs to generate candidate solutions to solve optimization problems with a single objective and multiple constraints.

Table I shows a comparison of INFORM with other synthesis methodologies. We only consider the recent BO methods from [17] that solve CMOO. EA is sample-inefficient and often requires repeated simulations to obtain a system configuration whose response meets the constraints and achieves the best value of the objectives. Other methods are sample-efficient, although the complexity of BO is cubic in the number of evaluations. All methods except ASSENT can handle multiple objectives and constraints. Step 2 of ASSENT uses an inverse design methodology to perform targeted simulations, making it amenable to varying designer specifications. INFORM uses three new inverse design methodologies and their combination to simultaneously generate multiple candidate solutions in comparison to ASSENT that generates a single candidate solution at each iteration.

III. Background

This section discusses the preliminary material that our work builds on. First, we discuss the formulation of system design as a CMOO problem. We then provide a brief overview of GA and its use in system optimization. Next, we discuss material that forms the basis for our inverse system design methodology using a GMM and NN-Ver.

TABLE I: Comparison of synthesis methodologies with INFORM

	BO	EA	ASSENT	INFORM
Sample-efficient	✓	×	✓	✓
Multiple objectives and constraints	✓	✓	×	✓
Inverse design	×	×	✓	✓
Multiple candidate solutions	✓	✓	×	✓
Dynamic specifications	×	×	✓	✓

A. CMOO Formulation for System Design

System design requires choosing component values from a constrained design space to find system configurations that achieve the best objective values while satisfying the defined constraints. Solutions to the CMOO problem constitute a set of non-dominated solutions that lie on the *Pareto front* [1]. CMOO for system design can be formulated as follows:

$$\begin{aligned}\underset{\boldsymbol{x}}{\text{minimize}} \quad & f_m(\boldsymbol{x}),\ m=1,2,\ldots,M \\ \text{subject to} \quad & g_j(\boldsymbol{x}) \geq 0, j=1,2,\ldots,J \\ & h_k(\boldsymbol{x})=0, k=1,2,\ldots,K \\ & x_i^L \leq x_i \leq x_i^U, i=1,2,\ldots,n\end{aligned} \tag{1}$$

where $\boldsymbol{x}$ is the design space over all possible component values and $f_m(\boldsymbol{x})$ is the m^{th} objective. The system must satisfy J inequality constraints given by $g_j(\boldsymbol{x})$ and K equality constraints given by $h_k(\boldsymbol{x})$. For example, when designing a Lunar lander [18], the goal is to maximize the reward determined by the position at which the module lands, minimize the fuel consumption, subject to constraints on the time taken to land, and a successful mission. The range of values of available components constrains the design space. The lower (upper) bound of value x_i of component i is labeled as x_i^L (x_i^U).

B. Genetic Algorithm

GA helps explore a large design space and is better suited at avoiding local minima than gradient-based optimization techniques. A set of candidate solutions comprising a population is evolved over multiple generations to improve the value of the objective function. A chromosome represents a candidate solution. It includes a sequence of genes that indicates the value of each component. A simulator evaluates the chromosomes to determine their objective values and if they meet the constraints. A subset of the best-performing chromosomes is selected to produce children that become candidate solutions for the next generation. The children are generated using crossover and mutation. The genes of two parents are exchanged at a selected crossover point to produce children. The value of a gene is changed with a given probability to effect mutation. The best-performing solutions among the current generation and the children are retained to form the next generation. This procedure continues until stopping criteria are met. Upon termination, the solution to the CMOO problem is the set of non-dominated solutions.

C. Inverse Design Preliminaries

We present a brief outline of GMM and NN-Ver.: two of the three inverse design methods used in INFORM.

1) GMM: A GMM assumes that the data probability distribution can be approximated by a finite number of Gaussian distributions. The probability density function (PDF) of a GMM with M components is denoted by $p(\mathbf{x})$:

$$p(\mathbf{x}) = \sum_{m=1}^{M} \pi_m \mathcal{N}(\mathbf{x};\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m). \tag{2}$$

In Eq. (2), $\mathbf{x}$ represents the features, and π_m, $\boldsymbol{\mu}_m$, and $\boldsymbol{\Sigma}_m$ denote the weight, mean, and variance of the m^{th} component, respectively. In scenarios where some of the features of an example are known, the unknown features can be estimated using the following equations [19]:

$$\begin{aligned} p(\mathbf{x}_\mathcal{P}) &= \sum_{m=1}^{M} \pi_m \mathcal{N}(\mathbf{x}_\mathcal{P};\boldsymbol{\mu}_{m,\mathcal{P}},\boldsymbol{\Sigma}_{m,\mathcal{PP}}) \\ p(\mathbf{x}_\mathcal{M} \mid \mathbf{x}_\mathcal{P}) &= \sum_{m=1}^{M} \pi_{m,\mathcal{M}|\mathcal{P}} \mathcal{N}\left(\mathbf{x}_\mathcal{M};\boldsymbol{\mu}_{m,\mathcal{M}|\mathcal{P}},\boldsymbol{\Sigma}_{m,\mathcal{M}|\mathcal{P}}\right) \\ \pi_{m,\mathcal{M}|\mathcal{P}} &= \pi_m \mathcal{N}(\mathbf{x}_\mathcal{P};\boldsymbol{\mu}_{m,\mathcal{P}},\boldsymbol{\Sigma}_{m,\mathcal{PP}})/p(\mathbf{x}_\mathcal{P}) \\ \boldsymbol{\mu}_{m,\mathcal{M}|\mathcal{P}} &= \boldsymbol{\mu}_{m,\mathcal{M}} + \boldsymbol{\Sigma}^T_{m,\mathcal{PM}}\boldsymbol{\Sigma}^{-1}_{m,\mathcal{PP}}(\mathbf{x}_\mathcal{P} - \boldsymbol{\mu}_{m,\mathcal{P}}) \\ \boldsymbol{\Sigma}_{m,\mathcal{M}|\mathcal{P}} &= \boldsymbol{\Sigma}_{m,\mathcal{MM}} - \boldsymbol{\Sigma}^T_{m,\mathcal{PM}}\boldsymbol{\Sigma}^{-1}_{m,\mathcal{PP}}\boldsymbol{\Sigma}_{m,\mathcal{PM}} \\ \mathbf{f}(\mathbf{x}_\mathcal{P}) &= \mathrm{E}\{\mathbf{x}_\mathcal{M} \mid \mathbf{x}_\mathcal{P}\} = \sum_{m=1}^{M} \pi_{m,\mathcal{M}|\mathcal{P}}(\mathbf{x}_\mathcal{P})\,\boldsymbol{\mu}_{m,\mathcal{M}|\mathcal{P}}(\mathbf{x}_\mathcal{P}). \end{aligned} \tag{3}$$

In Eq. (3), the present and missing features of an example are denoted by $\mathbf{x}_\mathcal{P}$ and $\mathbf{x}_\mathcal{M}$, respectively. $\boldsymbol{\mu}_{m,\mathcal{P}}$ and $\boldsymbol{\mu}_{m,\mathcal{M}}$ are computed by indexing $\boldsymbol{\mu}_m$ corresponding to the locations of the present and missing features of the given example. $\boldsymbol{\Sigma}_{m,\mathcal{PP}}$, $\boldsymbol{\Sigma}_{m,\mathcal{PM}}$, and $\boldsymbol{\Sigma}_{m,\mathcal{MM}}$ are computed analogously by indexing on $\boldsymbol{\Sigma}_m$. The PDFs of the present features and missing features conditioned on the present features are denoted by $p(\mathbf{x}_\mathcal{P})$ and $p(\mathbf{x}_\mathcal{M} \mid \mathbf{x}_\mathcal{P})$, respectively. The estimated value of the missing features, $\mathbf{f}(\mathbf{x}_\mathcal{P})$, is obtained from the expected value of the missing features conditioned on the present features. In the context of system design, $\mathbf{x}_\mathcal{M}$ corresponds to the unknown component values and $\mathbf{x}_\mathcal{P}$ corresponds to desired objectives/constraints.

2) NN-Verifier: An NN-Ver. determines whether there is an input corresponding to a given output of an NN. We use NSVerify [20], a framework for verifying an NN with ReLU activation by representing the hidden neurons with constraints, as follows:

$$\begin{aligned} C_i = \Big\{ & \bar{x}_j^{(i)} \geq W_j^{(i)}\bar{x}^{(i-1)} + b_j^{(i)}, \\ & \bar{x}_j^{(i)} \leq W_j^{(i)}\bar{x}^{(i-1)} + b_j^{(i)} + Q\bar{\delta}_j^{(i)}, \\ & \bar{x}_j^{(i)} \geq 0, \bar{x}_j^{(i)} \leq Q\left(1-\bar{\delta}_j^{(i)}\right), j=1,\ldots,L^{(i)} \Big\}. \end{aligned} \tag{4}$$

In Eq. (4), $\forall i,j$, $\bar{x}_j^{(i)}$ corresponds to the j^{th} neuron in the i^{th} layer, $L^{(i)}$ is the number of neurons in the i^{th} layer, $W_j^{(i)}$ represents weights that determine the input to $\bar{x}_j^{(i)}$, $\bar{x}^{(i-1)}$ represents outputs from the $(i-1)^{th}$ layer, $b_j^{(i)}$ is the bias for neuron $\bar{x}_j^{(i)}$, Q is larger than the largest possible magnitude of $W_j^{(i)}\bar{x}^{(i-1)} + b_j^{(i)}$, and $\bar{\delta}_j^{(i)}$ is defined as follows:

$$\bar{\delta}_j^{(i)} \triangleq \begin{cases} 0 \text{ if } \bar{x}_j^{(i)} > 0 \\ 1 \text{ otherwise} \end{cases} \tag{5}$$

The union of C_i's in Eq. (4) represents the constraints imposed by the hidden neurons of the NN. CNMA [15] and ASSENT [16] use the MILP formulation of an NN to synthesize systems with a single objective and multiple constraints. INFORM can handle CMOO problems and is, hence, more general.

Besides NN-Ver. and GMM, we also use an NN in INFORM. Rather than optimizing an objective function through gradient-based or gradient-free techniques, inverse design requires specifying the desired system performance. Inverse mapping enables a designer to drive the system to different performance metrics without having to start the optimization from scratch for each specification.

IV. Motivation

We provide a preview of the working of INFORM through the design of an inverted cartpole [18]. Sections V and VI provide further details on the experimental setup. The goal is to minimize the time ($t_{upright}$) to make the pole upright and minimize the maximum cart displacement (max_{disp}) for a fixed iterative linear quadratic regulator. The component values that need to be determined are cart mass, pole length, and magnitude of force applied to the cart.

A. Step 1: Obtaining Non-dominated Solutions

In Step 1, we obtain a non-dominated set of solutions by injecting candidate solutions within GA using the following inverse design approaches: (1) NN-Ver., (2) GMM, (3) NN, and their combinations, resulting in seven inverse design methods. We start injecting individuals using the inverse design methodology upon observing no improvement in the extreme values of the objective function for two consecutive generations. We terminate Step 1 when (1) we do not observe any improvement in the extreme values of the objective function for 10 generations after the tenth generation, or (2) Step 1 has run for seven hours on an Intel Xeon processor with 64 GB of DRAM. Fig. 1(a) shows, using black circles, the non-dominated front (NDF) obtained in Step 1 using NSGA-II and the inverse design methods. The x-axis shows the time taken to make the cartpole upright and the y-axis shows the cart's displacement. Injecting candidate solutions using the inverse design method(s) in conjunction with GA yields better solutions than the solutions obtained using NSGA-II alone. Fig. 1(b) plots *hypervolume* vs. *#generations*. Inverse designs obtained using NN-Ver.+GMM yield the highest hypervolume, indicating the benefits of injecting candidate solutions using inverse design methods.

B. Step 2: Dominating the Selected Solutions from Step 1

In Step 2, we select solutions from the Step 1 NDF to enhance their performance. Fig. 1(c) shows two selected solutions, labeled (1) and (2), with objective values of (7.20s,0.579m) and (11.25s,0.287m), respectively. We aim to lower the time to make the cartpole upright while achieving a displacement at least as good as that of the selected solution. First, we generate and simulate 20 candidate solutions within $\pm 10\%$ of the component values of the selected solution. We choose 100 simulations from Step 1 with performance closest to the selected solution to train the inverse design surrogate models. Fig. 2(a) shows ten desired objectives/constraints around the solution selected in Step 1. We dynamically change the solution around which we generate the desired objectives/constraints upon observing an improvement in the time objective, as in Fig. 2(b) that shows samples around the improved solution (9.0s,0.166m).

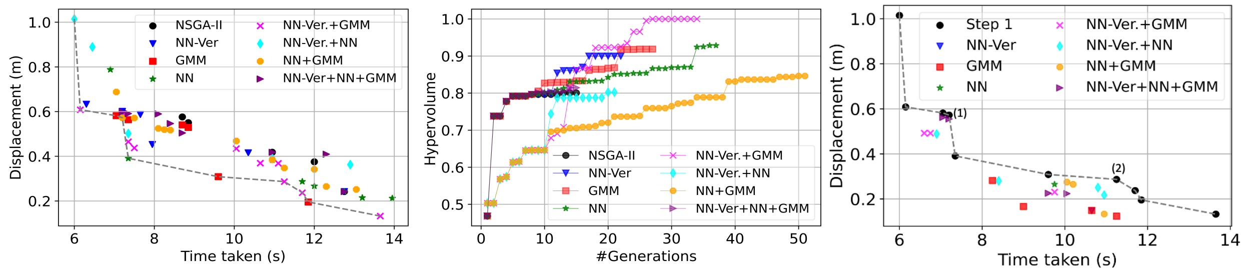
V. Synthesis Methodology

INFORM uses a two-step process to solve CMOO problems. Its high-level view is shown in Fig. 3. In Step 1, we modify a GA and inject candidate solutions using inverse design methodologies. The designer selects a solution from the NDF to further improve the value of the objective functions in Step 2. At the end of Step 2, INFORM produces solutions that dominate the solution selected from Step 1.

A. Step 1: Obtaining Non-dominated Solutions

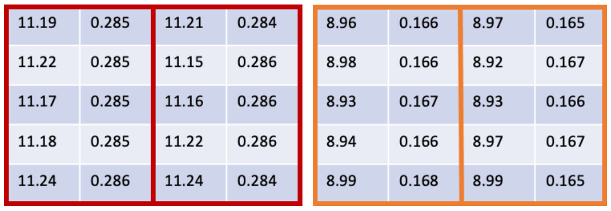
In Step 1, we initialize the first generation of GA within the design search space of the component values. Then, we evolve these candidate solutions using a GA until saturation. After that, we inject candidate solutions using inverse design methodologies and include the solutions in the GA pool along with the candidate solutions generated using crossover and mutation. We evolve these solutions until we meet termination criteria. Finally, we return the set of non-dominated solutions.

Algorithm 1 describes Step 1 in detail. We initialize the first GA generation with *P* candidate solutions using *Latin hypercube sampling (LHS)* [21] based on points within the design space *R* of component values. The initial phase of Step 1 is the same as that in NSGA-II [2]. Note that any CMOO exploration tool like NSGA-III [10] could also be used instead of NSGA-II. We simulate all individuals in the generation to compute the corresponding objective values. We store the simulation inputs and the corresponding outputs in a buffer *B*. We use the buffer as a lookup table when repeating a simulation to minimize the total number of simulations. We use NSGA-II to rank the individuals in a generation. Next, we use tournament selection to create a mating pool of size *P*. We then use *crossover* and *mutation* with a probability of *cross* and *mut*, respectively, to create *P* children. We use NSGA-II to select *P* individuals from the *2P* individuals for the next generation. If the evolution saturates (indicated by the *switchGA* flag), we begin injecting candidate solutions using one of the seven inverse design methods *invDesignMethod*. We set the *switchGA* flag to *True* when the extreme values of the objective function do not change for a few generations. Using extreme values helps discover the approximate range for each objective value. We train the surrogate models used for inverse designs after each generation using past simulations. We generate *numInvDesign* candidate solutions using *invDesignMethod* and add them to the *P* candidate solutions generated using GA. At each iteration, we select *numInvDesign* randomly. Random selection helps generate more diverse individuals, thus avoiding or delaying performance saturation. We only retain feasible solutions of the MILP when using an NN-Ver. for inverse design. We use all the candidate solutions generated using the inverse design method and some candidate solutions generated using crossover and mutation to obtain *P* candidate solutions for the next generation. We terminate Step 1 when we meet the stopping criteria *stop*.



(a) (b) (c)

Fig. 1: Cartpole optimization: (a) NDFs obtained using NSGA-II and the seven inverse design methods. Dashed grey line shows the NDF of the union of all solutions obtained in Step 1. (b) Comparison of hypervolume using the eight methods in Step 1. (c) Objective value improvement using Step 2 by dominating two solutions (labeled (1) and (2)) obtained in Step 1.



11.19	0.285	11.21	0.284
11.22	0.285	11.15	0.286
11.17	0.285	11.16	0.286
11.18	0.285	11.22	0.286
11.24	0.286	11.24	0.284

(a)

8.96	0.166	8.97	0.165
8.98	0.166	8.92	0.167
8.93	0.167	8.93	0.166
8.94	0.166	8.97	0.167
8.99	0.168	8.99	0.165

(b)

Fig. 2: Generation of desired objectives/constraints around: (a) design (11.25s,0.287m) selected after Step 1, and (b) after observing an improvement in objective value to (9.0s,0.166m) in Step 2. Note that our target is to obtain time and displacement that are at least as good as the solution chosen from Step 1. Therefore, we clip the displacement if it exceeds the displacement of the solution from Step 1.

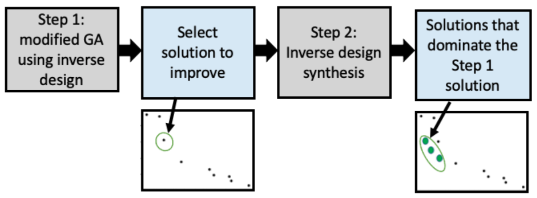

Step 1: modified GA using inverse design
Select solution to improve
Step 2: Inverse design synthesis
Solutions that dominate the Step 1 solution

Fig. 3: Overview of the two-step optimization process used in INFORM.

Algorithm 2 shows the procedure for generating desired objectives/constraints around design D. In CMOO problems, D depicts solutions that meet the constraints and attain the top ranks as determined by NSGA-II. If none (or fewer than the cardinality of D) of the solutions satisfy the constraints, we use the solutions with minimum constraint violation computed using Eq. (6).

$$\sum_i \frac{|\text{obs}_i - \text{con}_i|}{\text{con}_i} \mathbb{1}_{\{\text{obs}_i \lessgtr \text{con}_i\}}, \quad (6)$$

where con_i is the i^{th} constraint, obs_i is the observed value for the i^{th} constraint, and $\mathbb{1}_{\{\text{obs}_i \lessgtr \text{con}_i\}}$ is an indicator function that takes the value 1 in case of con_i violation (note that obs_i can be greater or less than con_i) and 0 otherwise. We determine the percentage improvement fr to be targeted in the objective values around the solution in D by generating a random number (fr) in the [lb_per,ub_per] range. Small values of fr lead to exploitation near the best solution. Higher values lead to exploration. We generate N desired objectives/constraints (*des_obj_ctr*) per

```
Algorithm 1 Step 1: Discovering non-dominated solutions
Input: P: population size; comp_range: range of each input
    component; stop: stopping criteria; switchGA: switching cri-
    terion for GA+inverse design; B: buffer to store simulations;
    mut: mutation probability; cross: crossover probability;
    invDesignMethod: inverse design choice(s): (1) NN-Ver.,
    (2) GMM, (3) NN, (4) NN-Ver.+GMM, (5) NN-Ver.+NN,
    (6) NN+GMM, (7) NN-Ver.+NN+GMM; numInvDesign:
    #inverse designs injected per generation.
    - Initialize 1st generation with P LHS solutions within
    comp_range
    while not stop do
        - Compute objectives for all P
        - Store simulation inputs/outputs in B
        - Rank P using NSGA-II
        - Use tournament selection to create mating population
        of size P
        - Use reproduction based on crossover and mutation to
        create P children
        if switchGA then
            - Update the surrogate used in invDesignMethod
            - Generate numInvDesign using invDesignMethod
            - Append numInvDesign candidate solutions to the
            ones generated using crossover and mutation
            - Discard some candidate solutions generated using
            crossover and mutation to retain only P candidate
            solutions
        end if
        - Select P from 2P members using NSGA-II
    end while
Output: All non-dominated solutions and buffer B
```

solution from D. At each successive call to Algorithm 2, we alternate between Sobol and LHS *point_type* to generate *des_obj_ctr* around the *candidate* in D. Sobol samples are distributed uniformly over a unit hypercube [22]. Alternating between Sobol sampling and LHS to generate *des_obj_ctr* and using a random value for fr helps avoid performance saturation. We clip *des_obj_ctr* to satisfy all constraints (*des_obj_ctr_clp*). The content of *des_obj_ctr_clp* is stored in the *des_obj_ctr_list* list.

Algorithm 3 shows NN-Ver.-based inverse design. We select

Algorithm 2 Generation of desired objectives/constraints

```
Input: D: solutions around which desired objectives/constraints des_obj_ctr are generated; lb_per, ub_per: lower and upper bound percent for generating des_obj_ctr; N: number of des_obj_ctr per design from D; point_type: LHS or Sobol samples.
  fr = random number in the [lb_per,ub_per] range
  des_obj_ctr_list = []
  for candidate in D do
    des_obj_ctr = Generate N point_type samples within fr around candidate solution
    des_obj_ctr_clp = clip des_obj_ctr_clp to satisfy the constraints
    Append des_obj_ctr_clp to des_obj_ctr_list
  end for
Output: des_obj_ctr_list
```

the top solutions (*top_sols*) from buffer B to train the NN surrogate model. We select all (*num_train*) the solutions that meet the constraints to train the surrogate model. If the number of solutions that meet the constraints is large, we clip *num_train* to lie in [lb train, ub train]. We scale component values and the corresponding desired objectives/constraints to [0,1]. We select the best NN architecture (*best_arch*) from the *NN_archs* choices for different NN architectures. Next, we use Eq. (4) to convert *best_arch* into its corresponding MILP using the desired objectives and constraints from *des_obj_ctr_list* generated using Algorithm 2. Since we have a different MILP for each desired *des_obj_ctr*, we use multiple cores to solve the MILP. Finally, we return *can_sols_NN*, which is the list of all feasible solutions obtained by solving the MILP.

Algorithm 3 NN-Verifier-based inverse design

```
Input: des_obj_ctr_list: list of desired objectives and constraints; B: stored buffer with simulation inputs and outputs; lb_train, ub_train: lower and upper bound on the number of solutions to train the surrogate model; NN_archs: search space of NN architectures; comp_range: range of each input component.
  num_train = random number in [lb_train,ub_train]
  num_train = min(num_train, buffer size B)
  top_sols = select the best num_train solutions from B
  best_arch = best NN architecture among NN_archs trained on top_sols
  can_sols_NN = []
  for des_obj_ctr in des_obj_ctr_list do
    Use Eq. (4) to convert best_arch into corresponding MILP
    Set output constraint of the MILP to des_obj_ctr and input constraint to comp_range
    sol_milp = MILP solution to determine the inputs
    if sol_milp is feasible then
      append sol_milp to can_sols_NN
    end if
  end for
Output: can_sols_NN
```

Fig. 4(a) shows the data used to train the NN surrogate model for NN-Ver.-based inverse design. The illustration assumes that each design has four components, three objectives/constraints, and *num_train* = N. Each row represents a training example. X_{ij} represents the j^{th} component of the i^{th} training instance. Similarly, Y_{ij} represents the j^{th} objective/constraint of the i^{th} training instance. Fig. 4(b) depicts *des_obj_ctr* using $\hat{Y}_{ij}$, where i is the output number and j is the j^{th} component of the i^{th} *des_obj_ctr*. The unknown value of the components for each *des_obj_ctr* is depicted by a '?'. We generate M candidate solutions using Algorithm 2.

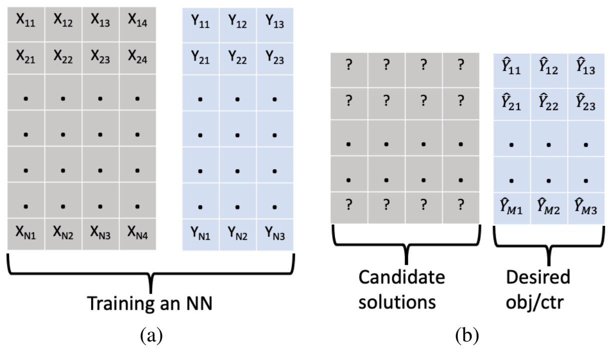


Fig. 4: Inverse design using NN-Ver.: (a) training dataset for the surrogate model, and (b) generating candidate solutions using *des_obj_ctr*.

Algorithm 4 shows GMM-based inverse design. The training setup is similar to that in Algorithm 3. We create the training dataset *X_Y* for GMM by concatenating the simulation inputs (i.e., component values) and its corresponding outputs (objectives/constraints). We determine the best GMM (*best_gmm*) from *mix_space* by varying the number of mixtures used for training. We use the GMM with the lowest Akaike information criterion (AIC) [23] score. We use $\mathbf{f}(\mathbf{x}_{\mathcal{P}})$ from Eq. (3) to determine the candidate solutions for each *des_obj_ctr* from *des_obj_ctr_list*. We clip the solutions to lie within the range of each component (*com_range*). We return the list of all the candidate solutions denoted by *can_sols_gmm*.

Fig. 5(a) illustrates how training is done with the GMM. In contrast to an NN-Ver.-based inverse design that treats the input components and corresponding objectives/constraints separately, we concatenate the two using past simulations to train the GMM. Fig. 5(b) shows how candidate solutions are generated. We use the expected value of the conditional distribution of the input, given *des_obj_ctr*, to determine the unknown component values shown by '?'.

Finally, we describe NN-based inverse design. We invert NN architectures used in NN-Ver. to determine the search space of the NNs. For example, if we use NN with [(40,20,8)] neurons in the hidden layers when using NN-Ver.-based inverse design, we replace this architecture with [(8,20,40)]. The inverted architecture maps the simulation outputs to the corresponding inputs. We use *num_train* data instances to determine the best NN architecture and use the NN to predict the component values for each *des_obj_ctr*. Like GMM-based inverse design, we clip the candidate solutions to lie within the range of each component (*comp_range*). We then return the list of all the candidate solutions.

Algorithm 4 GMM-based inverse design

```
Input: des_obj_ctr_list: list of desired objectives and constraints; B: stored buffer of simulation inputs and outputs; lb_train, ub_train: lower and upper bound on the number of samples to train the surrogate model; mix_space: search space for number of mixtures to train the GMM; comp_range: range of each input component.
    num_train = random number in [lb_train,ub_train]
    num_train = min(num_train, buffer size B)
    top_sols = select the best num_train solutions from B
    X_Y = [X;Y], concatenate input and output columns of top_sols
    best_gmm = best GMM from the mix_space trained on X_Y
    can_sols_gmm = []
    for des_obj_ctr in des_obj_ctr_list do
        sol_gmm = use Eq. (3) to determine the expected value of the unknown input component $\mathbf{f}(\mathbf{x}_{\mathcal{P}})$ after setting the objective and constraint $(x_{\mathcal{P}})$ to des_obj_ctr_list
        sol_gmm_clip = clip sol_gmm to lie within comp_range
        append sol_gmm_clip to can_sols_gmm
    end for
Output: can_sols_gmm
```

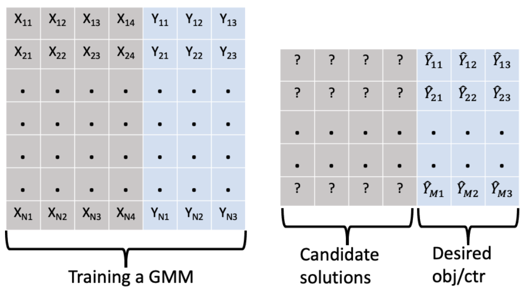


Fig. 5: Inverse design using GMM: (a) training the GMM with concatenated input and output data, and (b) generating candidate solutions using *des_obj_ctr*.

B. Step 2: Dominating Selected Solutions from Step 1

In Step 2, we use the inverse design method to improve the solution from Step 1. Fig. 6 shows the fine-tuning procedure. We generate and simulate *N_init_Step2* LHS samples within *config_sam_pr1* % around the component values (*config_sel*) corresponding to the solution chosen in Step 1. Note that in the case of costly simulations, an alternative approach would be to use some logged simulations from the buffer (*B*) in Step 1 to pre-train the surrogate model used for inverse design, as illustrated in the motivational example. We use the simulation inputs (i.e., component values) and corresponding outputs (i.e., objectives/constraints) to train the surrogate model used for inverse design. We use Algorithm 2 to generate *N_inv des_obj_ctr* with the aim of improving the value of the objective function by up to *pr*%. We generate candidate solutions (*can_sols*) using the same inverse design methods as in Step 1. We simulate *can_sols* and update the component values (*best_config*) corresponding to the best objective value (*best_obj_ctr*) on observing a better solution. We use *pr1* as the improvement percentage whenever we observe an improvement in the value of the objective function. If we do not observe an improvement after *sam_th1* iterations of the above steps, we lower the improvement percent for *des_obj_ctr* to *pr2*. If no improvement is observed even after *sam_th2* iterations, we generate and simulate *N_invSat* LHS samples within $\pm$*config_sam_pr2* % of *best_config*. We repeat this process until we meet the stopping criteria (*stop_step2*) for Step 2. Finally, we return the NDF of the solutions that dominate the solution chosen from Step 1.

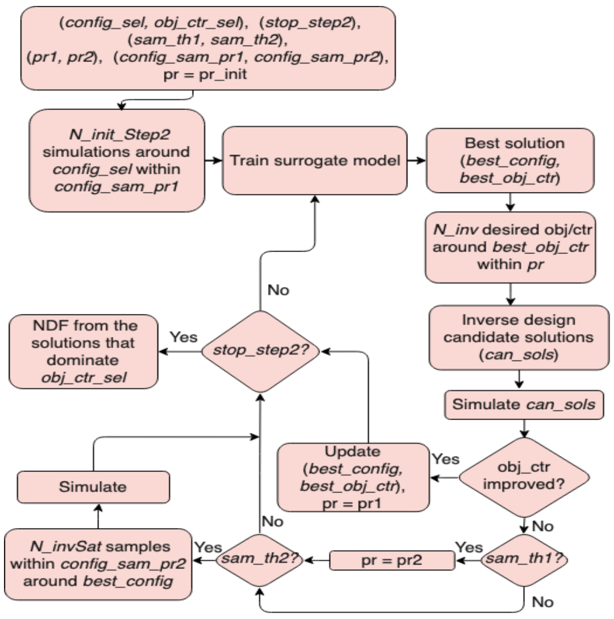

(config_sel, obj_ctr_sel), (stop_step2), (sam_th1, sam_th2), (pr1, pr2), (config_sam_pr1, config_sam_pr2), pr = pr_init
N_init_Step2 simulations around config_sel within config_sam_pr1
Train surrogate model
Best solution (best_config, best_obj_ctr)
N_inv desired obj/ctr around best_obj_ctr within pr
Inverse design candidate solutions (can_sols)
Simulate can_sols
obj_ctr improved?
Yes
No
Update (best_config, best_obj_ctr), pr = pr1
sam_th1?
Yes
No
pr = pr2
sam_th2?
Yes
No
N_invSat samples within config_sam_pr2 around best_config
Simulate
stop_step2?
Yes
No
NDF from the solutions that dominate obj_ctr_sel

Fig. 6: Fine-tuning using Step 2

VI. Experimental Results

In this section, we use INFORM for component selection for Lunar Lander [18] and transimpedance amplifiers [5]. We implement INFORM using Keras [24], Scikit-learn [25], Gurobi [26], and PyGMO [27]. The simulations are performed on an Intel Xeon processor with 64 GB of DRAM.

A. Lunar Lander

We use the Lunar Lander benchmark from OpenAI/Gym [18] with the same setup as in [15]. The goal is to design the controller, determine the module's initial position, and optimize for two objectives subject to two constraints. The two objectives are: (1) maximize the reward, and (2) minimize the fuel used. The two constraints are (1) the time taken to land $\leq$ 10 units, and (2) the mission must be successful. We use the following objectives in GA:

- Fuel used in the mission.
- Time taken to land. If the time is less than or equal to 10 units, we set this objective value to 0 as we only need to satisfy the time constraint.
- Flag indicating whether a mission is successful. As we solve a minimization problem with GA, we flip the flag to take the value of 0 for success and 1 otherwise.
- Reward achieved in the mission. We negate the reward sign and convert it into a minimization problem.

In the event of a failed mission, we set the fuel, time, and reward objectives to a large number to indicate failure.

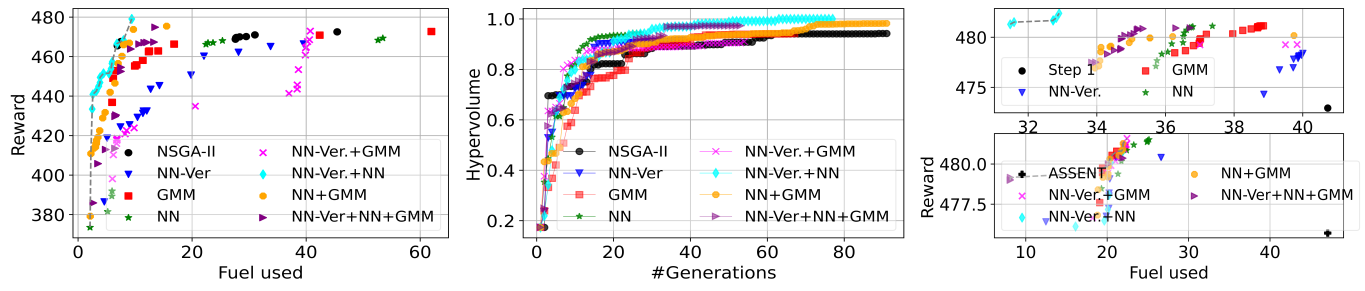
We run Step 1 for about seven hours (allowing the last generation to complete even if it exceeds the allotted time slightly) or until performance saturation. Lunar lander simulations are fast. Hence, the saturation criterion generally leads to the termination of Step 1. We start injecting candidates into GA using inverse designs if the extreme values of the objectives do not improve for two consecutive generations after the second generation. We assume saturation if the extreme value of the objective function remains unchanged for ten generations after the tenth generation. This saturation criterion worked well for all designs considered in this article.

We use a tournament size of 10, a mutation rate of 0.1, and a crossover probability of 0.9 for the GA. We run Step 1 using NSGA-II alone, and NSGA-II with injections using the seven inverse design methods. When using NN-Ver., we select an NN with one of the following architectures (*NN_archs*): [(40,20,8), (30), (15), (100), (50), (200), (20,20,8), (300), (400), (500), (600)] (same as in Step 2 of ASSENT [16]), where the tuples represent the number of neurons in the hidden layer(s). When using NN, we invert the NN architectures choices from that used in NN-Ver.: [(8,20,40), (30), (15), (100), (50), (200), (8,20,20), (300), (400), (500), (600)]. We use GMM and a *Multi-Layer Perceptron Regressor* from the Scikit-learn [25] package, an initial learning rate of 0.0001, *adaptive* learning, and a *maximum iteration count* of 100,000 to train the NN. We set other parameters to their default values. When using GMM, we vary *#mixtures* in the [1,*min*(200,*length*(train))] range in steps of 2. Here, *length*(train) denotes the number of training data instances. We limit the maximum number of mixtures and skip alternate component choices to speed up training. We update the choice of the surrogate model (GMM and NN) after each generation. We randomly select *num_train* training instances in the [1000,2000] range to train the surrogate model. When using only GMM, NN-Ver. or NN, we randomly inject inverse design candidates in the α = [10,96] range. When using NN-Ver.+GMM, NN-Ver.+NN, or NN+GMM, we set α = [10,48], and when using NN-Ver.+NN+GMM, we set α = [10,32] for each inverse design method. We randomly select top candidates in the D = [2,λ] range, where $\lambda = min(\alpha/2, 10)$. We inject the same number of candidates (N = $\lfloor \alpha/D) \rfloor$) around each selected candidate. We randomly select the percentage improvement required (*fr*) from the [1,10]% range. We clip *des_obj_ctr* in case of a constraint violation. For example, if *des_obj_ctr* is [8,11,1,400], we change it to [8,10,1,400] to meet the constraint on the time taken. The items included in *des_obj_ctr* are fuel used, time taken, success, and reward, respectively.

Fig. 7(a) shows the Step 1 NDF for the eight cases. We notice that using GA in conjunction with inverse design yields better performance. We attribute the improvement in performance to targeted candidate solutions generated using inverse designs. Fig. 7(b) plots *hypervolume* vs. *#generations*. We scale the two objectives to lie within [0,1] and use (1.1, 1.1) as the reference point. Hypervolume computation is done using the setup given in the PyMOO [28] documentation. We normalize the hypervolume to the maximum hypervolume obtained across the eight methods. We observe that injecting solutions based on inverse designs in conjunction with GA yields better hypervolume in comparison to NSGA-II alone, with NN-Ver.+NN having the highest hypervolume. For a fair comparison, we run NSGA-II until the maximum #generations attained with injections among the seven inverse designs methods (NN+GMM in this case). The number of simulations and time (sim. + algorithm) needed for each synthesis methodology are as follows: (1) GA: 8,132 sim., 293+90s; (2) NN-Ver.: 3,608 sim., 682+6,172s; (3) GMM: 4,887 sim., 180+586s; (4) NN: 2,631 sim., 99+30,600s; (5) NN-Ver.+GMM: 4,543 sim., 409+6,648s; (6) NN-Ver.+NN: 7,439sim., 962+13,899s; (7) NN+GMM: 7,829 sim., 294+10,545s; (8) NN-Ver.+NN+GMM: 4,724 sim., 494+7,158s. GA takes the least time as crossover and mutation are fast operations. Solving MILP makes NN-Ver.-based inverse design slow. Training the GMM surrogate and solving Eq. (2) is a relatively fast operation.

In Step 2, we improve a solution from the NDF corresponding to the highest reward obtained with NN-Ver.+GMM injections in Step 1. When generating *des_obj_ctr*, we try to improve the value of the reward and set the fuel used corresponding to the selected solution as a constraint. Note that we only apply the fuel constraint when *des_obj_ctr* (generated by sampling within $pr\%$ of the selected solution) exceeds the fuel consumed by the Step 1 solution. We keep the other constraints the same as in Step 1. We use the seven inverse design techniques to obtain solutions that dominate the Step 1 solution. We use the same search space for the surrogate models, that is, NN architecture choices and number of GMM mixtures used in Step 1. As the lunar lander simulations are fast, we only use the simulations that meet all the constraints to train the surrogate model.

We first generate *N_init_Step2* = 100 candidate solutions using LHS within *config_sam_pr1* $\%$ = $\pm 10\%$ of the component values of the configuration (*config_sel*) selected from Step 1. We use these simulation inputs and the corresponding outputs to train the surrogate model. We stop (*stop_step2*) Step 2 if: (1) there is no improvement in the value of the objective function (reward in this case) for 25 iterations, or (2) Step 2 has run for 7 hours. The (*best_config*, *best_obj_ctr*) corresponds to the solution with the highest reward and a fuel consumption that is at least as good as the solution chosen in Step 1. Here, *best_config* refers to the component values of the configuration that yields the best value of the objective function. In each iteration of Step 2, we randomly generate candidate solutions in the *N_inv* = [10,50] range for each inverse design technique. In the first iteration of Step 2, we set *pr* = *pr_init* to (5,10)$\%$. We change *pr* = *pr1* to (1,5)$\%$ if we do not observe an improvement in the value of the objective function, and further change *pr* = *pr2* to (1,2)$\%$ if we do not observe an improvement for *sam_th1* = 5 iterations of Step 2. Upon observing an improvement, we reset *pr* = *pr1* to (1,5)$\%$. If we do not observe an improvement for *sam_th2* = 10 iterations, we generate *N_invSat*=100 LHS samples within *config_sam_pr2* $\%$ = 1$\%$ of the best system configuration (*best_config*). A lower value of *pr* helps in exploitation, while a higher value helps in exploration to rapidly improve the value of the objective function. Unless otherwise specified, we use the same setup



(a) (b) (c)

Fig. 7: Lunar Lander optimization: (a) NDFs obtained in Step 1. (b) Comparison of hypervolume using the eight methods in Step 1. (c) Dominating the Step 1 (top) and ASSENT solution (bottom) shown in black. Dashed-line shows the combined NDF obtained using the different synthesis methodologies.

for Step 1 and Step 2 in all experiments.

The top row of Fig. 7(c) shows the solution selected from Step 1 in black. We obtain solutions that dominate the selected solutions using all inverse design techniques. We observe that the highest reward (482.4) is obtained using NN-Ver.+NN-based inverse design. The time and number of simulations required in Step 2 corresponding to the selected solution from Step 1 are (1) NN-Ver.: 5,689 sim., 9.3+276.3 min; (2) GMM: 12,767 sim., 5.5+415.9 min; (3) NN: 6,132 sim., 2.9+350.9 min; (4) NN-Ver.+GMM: 9,890 sim., 8.3+415.5 min; (5) NN-Ver.+NN: 3,378 sim., 1.73+155.9 min; (6) NN+GMM: 3,846 sim., 1.9+125.5 min; (7) NN-Ver.+NN +GMM: 7,862sim., 21.8+405.9 min. Next, we illustrate Step 2's flexibility by improving an intermediate (not from the NDF) solution from Step 1 obtained using NN-Ver.+NN-based injections. We aim to enhance its performance while ensuring that it dominates the ASSENT [16] solution. We modify the time constraint to 3.5 from 10.0 to obtain solutions with lower time than that obtained using ASSENT. We show the ASSENT solution using a black '+' in the bottom row of Fig. 7(c). The time and number of simulations required in Step 2 in this case are (1) NN-Ver.: 6,687 sim., 9.3+286.1 min; (2) GMM: 6,977 sim., 5.5+176.3 min; (3) NN: 3,155 sim., 2.0+154.8 min; (4) NN-Ver.+GMM: 4,120 sim., 3.2+128.3 min; (5) NN-Ver.+NN: 6,259 sim., 10.1+277.7 min; (6) NN+GMM: 8,808 sim., 7.6+332.5 min; (7) NN-Ver.+NN+GMM: 7,890 sim., 8.5+280.1 min.

Table II compares results from INFORM with results from ASSENT [16] for Lunar Lander. The top half of the table shows a solution obtained using NN-Ver.+GMM injections in Step 1. Using NN-Ver.+NN-based inverse design in Step 2 yields solutions that dominate the Step 1 solution in two different runs. The bottom half of the table shows results of using Step 2 to enhance the Step 1 solution so that it further dominates the ASSENT [16] solution. NN-Ver.+NN+GMM-based inverse design yields a solution that consumes 6× lower fuel and a comparable performance in other metrics. NN-Ver.+GMM-based inverse design yields a solution that has 2× lower fuel and a 1.2% higher reward than the ASSENT solution.

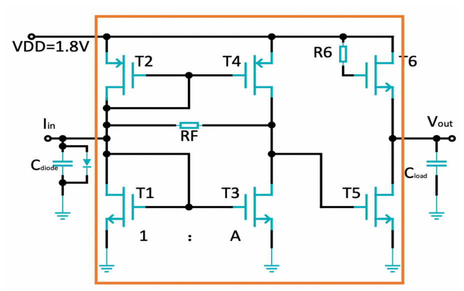
B. Two-stage Transimpedance Amplifier

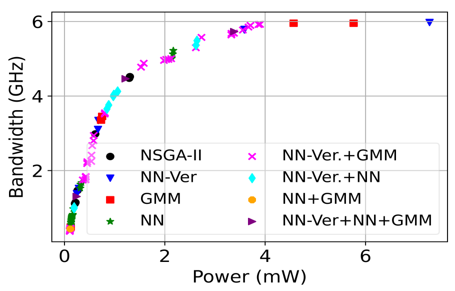
We design a two-stage transimpedance amplifier using the topology from [5]. The designer needs to determine the width of all MOSFETs and the two resistors shown in Fig. 8(a). The search space for the width of each MOSFET is in the $[0.18, 80]$ μm and the resistor in the $[50, 5\text{k}]$ Ω range. When running GA, we use the same three objectives as used in ASSENT [16]. The three objectives are:

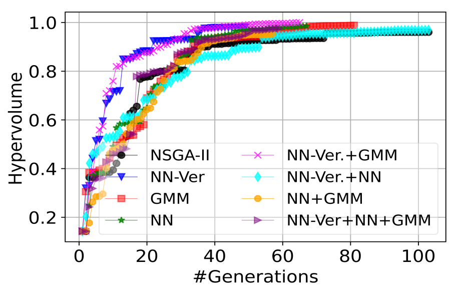
TABLE II: Comparing results from INFORM (two runs separated by a '/') with those from ASSENT for Lunar Lander.

Method	# Sim.	sim.+alg. time	fuel	time $\leq$ 10	reward
ASSENT [16]	1,983	54,042s	47.09	3.50	475.7
Step 1(NN-Ver.+GMM)	4,543	409+6,648s	40.73	3.44	472.9
Step 2(NN-Ver. +NN)	3,378/ 6,780	104+9,353/ 870+21,633s	32.90/ 32.80	3.54/ 3.12	482.4/ 480.8
Step 1(NN-Ver.+NN)	7,439	962+13,899s	22.0	3.26	477.4
Step 2(NN-Ver. +NN+GMM)	7,890	507+16,804s	8.10	2.32	479.0
Step 2(NN -Ver.+GMM)	4,120/ 7,532	190+7,700/ 684+13,653s	22.44/ 23.93	3.34/ 3.22	481.6/ 480.8

1) *Bandwidth*: We capture all the metrics (gain, peaking, and bandwidth) corresponding to the frequency response into one objective. In addition, we include the operating mode of the MOSFET in this objective as well. We obtain the objectives corresponding to the frequency response as follows. (1) We set the target gain to 60 dB with a cutoff frequency of 6.1 GHz. We define the first sub-objective as the difference in the target response and the observed response. In order to focus more on the response in the passband, we add an additional term to the difference. This additional term is 40 times the difference between the target and the observed response in the passband [16]. (2) We penalize the gain by $\alpha = 15$ determined by the fractional deviation of the difference in gain below 58.1 dBΩ, achieved by the RL-synthesized circuit in [5]. (3) We add a penalty of 15 based on the fractional deviation in peaking above 0.963 (RL circuit in [5]). (4) Similarly, we add a penalty of 15 if the observed bandwidth is below 5.81 GHz [16]. We also penalize the frequency response based on the operating region of the MOSFET. We give a reward of 1 for each MOSFET operating in the saturation region. We penalize by 2 (3) an operation in the linear (cutoff) region. We sum the rewards and penalties and divide by the number of MOSFETs in the circuit. We use these rewards and penalties to encourage MOSFETs to operate in the saturation region for the circuit to behave as an amplifier. We sum all the sub-objectives for the frequency response and scale it by the objective value of the human-designed circuit in [5].
2) *Noise*: We define the noise objective as the ratio of the







(a) (b) (c)

Fig. 8: Two-stage transimpedence amplifier optimization: (a) Schematic adapted from [5]. We determine component values for devices within the orange box. (b) Non-dominated solutions for bandwidth vs. power. (c) Comparison of the hypervolume using the eight methods in Step 1.

measured noise and the noise corresponding to the RL circuit in [5] (19.2 $\mathrm{pA}/\sqrt{\mathrm{Hz}}$). If the measured noise is above 19.2 $\mathrm{pA}/\sqrt{\mathrm{Hz}}$, we apply a penalty of 15 determined by the fractional deviation above this value.

3) *Power*: We define the objective corresponding to power as the ratio of the measured power and the power achieved by the RL circuit in [5] (3.18 mW).

If the simulation is unsuccessful, we set the objective values to a very large number to indicate this fact. We use prior knowledge and assume a design is invalid if the power consumption is less than 0.1 μW.

As inverse design methods can handle multiple objectives and constraints, we consider all objectives and constraints independently, instead of clubbing them together. The goal of the inverse design is to meet the three constraints and optimize the three objectives. The constraints are: *noise* $\leq$ 19.3 $\mathrm{pA}/\sqrt{\mathrm{Hz}}$, *gain* $\geq$ 57.6 dB Ω, and *peaking* $\leq$ 1 dB. The objectives are: minimize *power*, minimize *area*, and maximize *bandwidth*. We clip *des_obj_ctr* corresponding to the objectives of the solution around which we generate samples to ensure that we improve all objective values simultaneously and satisfy the constraints. For example, if we sample around an objective and constraint with a value of [19.2 $\mathrm{pA}/\sqrt{\mathrm{Hz}}$, 57.7 dB$\Omega$, 0.95 dB, 6.0 mW, 20 $\mu\mathrm{m}^2$, 6.1 GHz] and generate a *des_obj_ctr* [19.3 $\mathrm{pA}/\sqrt{\mathrm{Hz}}$, 57.7 dB$\Omega$, 0.98 dB, 6.2 mW, 19.5$\mu\mathrm{m}^2$, 6.12GHz], we clip *des_obj_ctr* to [19.3 $\mathrm{pA}/\sqrt{\mathrm{Hz}}$, 57.7 dB$\Omega$, 0.98 dB, $\underline{6.0}$ mW, 19.5$\mu\mathrm{m}^2$, 6.12GHz]. Note that *des_obj_ctr* satisfies all constraints and is better or the same (power) in all objective values than the solution around which we generate the desired objectives and constraints. We found that clipping the objective values worked well on both the electrical circuit examples.

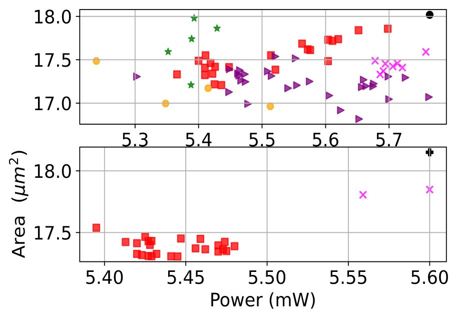
We select an NN with one of the following architectures: [(15), (30), (40,20,8)]. Fig. 8(b) shows the NDF for *bandwidth* vs. *power* obtained in Step 1. To enhance readability, we only plot the NDF of the union of all simulations obtained from the eight methods in Step 1. Injecting GA with candidate solutions using inverse designs leads to most of the solutions on the NDF. Fig. 8(c) plots *hypervolume* vs. *#generations*. We observe that inverse design injections using GMM and NN-Ver.+GMM achieve similar hypervolume, with the latter converging the fastest. The number of simulations and time (sim.+alg.) needed for each synthesis methodology are as follows: (1) GA: 7,873 sim., 58.5+0.8 min; (2) NN-Ver.:3,798 sim., 24.8+69.9 min; (3) GMM: 7,139 sim., 45.3+53.2 min; (4) NN: 5,637 sim., 36.4+31.6 min; (5) NN-Ver.+GMM: 5,745 sim., 37.1+135.7 min; (6) NN-Ver.+NN: 9,326 sim., 61.7+226.1 min; (7) NN+GMM: 5,353 sim., 34.4+60.3 min; (8) NN-Ver.+NN+GMM: 5,162 sim., 32.9+61.5 min.

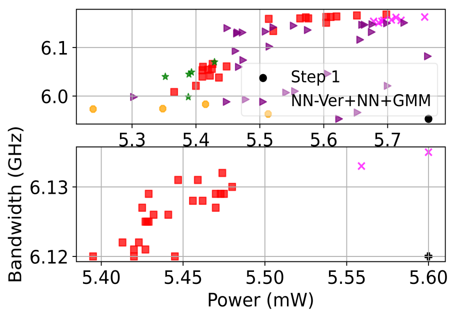
In Step 2, we use inverse designs to dominate the solution selected from the NDF in Step 1. We also generate a solution that is better or the same in all objectives and constraints compared to the solution obtained using ASSENT. We run Step 2 for a maximum of 10 (15) hours when generating a solution to dominate the Step 1 (ASSENT) solution. We improve the bandwidth value and restrict the area and power to the values of the solution chosen in Step 1. The top row of Fig. 9(a)-(c) shows the NDF of the solutions that dominate the Step 1 solution, depicted with a black circle. The number of simulations and the time (sim.+alg.) required in Step 2 corresponding to the solution from Step 1 are as follows: (1) NN-Ver.: 4,754sim., 36+565 min; (2) GMM: 8,018 sim., 53+548 min; (3) NN: 14,383 sim., 93+245 min; (4) NN-Ver.+GMM: 7,679 sim., 50+553 min; (5) NN-Ver.+NN: 9,989 sim., 80+523 min; (6) NN+GMM: 9,758 sim., 62+539 min; (7) NN-Ver.+NN+GMM: 7,315 sim., 59+551 min. Note that all inverse design methods yield solutions that dominate the Step 1 solution. However, the inverse design solutions using the five inverse design methods shown in the top row of Fig. 9 dominate the solutions obtained using the remaining two inverse design methods. Therefore, we do not plot the solutions obtained using all the seven methods.

In the second experiment, we demonstrate INFORM's flexibility by searching around the same solution selected in Step 1, but changing our goal to obtain solutions that dominate the solution obtained using ASSENT in all objectives and constraints. We set noise, gain, peaking, power, and area to constraints corresponding to the solution obtained using ASSENT and improve the bandwidth value. The bottom row of Fig. 9(a)-(c) shows the ASSENT solution using a black '**+**' sign.

In Table III[1], we compare the solution obtained using INFORM for two different runs with the solutions using other methodologies. We show the first solution that dominates the ASSENT solution in parentheses. The number outside the parentheses shows the solution obtained at the end of Step 2. INFORM achieves a speed-up of up to 24$\times$ (GMM second

[1]Gate area is shown only for designs for which this information is available. There was a problem in area calculation in [5] that was confirmed after contacting the authors.


(a)


(b)

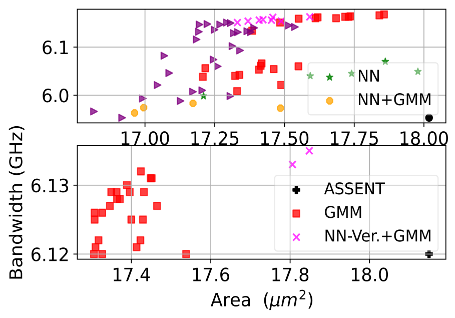

(c)

Fig. 9: Dominating the Step 1 (top) and ASSENT solution (bottom) shown in black for the two-stage transimpedance amplifier: (a) area vs. power, (b) bandwidth vs. power, and (c) bandwidth vs. area.

run) in synthesizing a solution with 3.4% lower area and a similar performance in other objectives/constraints as the ASSENT solution. In the bottom half of the table, we only use Step 2 to drive the human solution from [5] to further enhance its performance (objectives+constraints). INFORM yields solutions that have a higher bandwidth and are better or the same in other performance metrics.

C. Three-stage Transimpedance Amplifier

Let us now see how we can select the component values of the three-stage transimpedance amplifier shown in Fig. 10 [5]. The subcircuits in the blue and green dotted boxes are mirror images of each other. Hence, we select the component values of one subcircuit and mirror them to the other. We determine the values of 19 components: width/length of nine MOSFETs and a resistor *Rb*. The search space for the MOSFET's width is in the $[2, 30]$ μm, length in the $[1, 2.2]$ μm, and resistor in the $[50\text{k}, 500\text{k}]$ Ω range [16]. As the technology permits length to be an integer multiple of 0.2 μm, we round the length to the nearest 0.2 units.

We use the following objectives for the GA [16]:

1) *Bandwidth*: As in Sec. VI-B, we capture the metrics corresponding to the frequency response into one objective. We set the target gain for the half circuit to be 85 dB with a cutoff frequency of 90 MHz. We use the same technique (with the modified gain and cutoff frequency), as used in Sec. VI-B, to compute the first sub-objective to capture the frequency response. We also use the same method to compute the sub-objective corresponding to the operating region of the MOSFET, except using a penalty of 5 (7) for MOSFET operation in linear (cutoff) region [16]. The third sub-objective is a penalty of 15 for fractional deviation in gain below 80 dB for the half circuit in Fig. 10.
2) *Area*: We take the ratio of the area of the circuit synthesized in Step 1 using GA and the area of the human-synthesized circuit from [5].
3) *Power*: We take the ratio of the measured power to the power consumed by the human-synthesized circuit (1.37 mW) from [5]. We use a penalty of 15 for fractional deviation in power above 1.37 mW.

We generate candidate solutions using the inverse design methods by setting *des_obj_ctr* to maximize *bandwidth*, minimize *power*, and minimize *area* subject to the constraint on *gain* being greater than 80 dB.

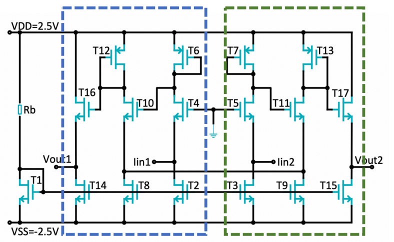


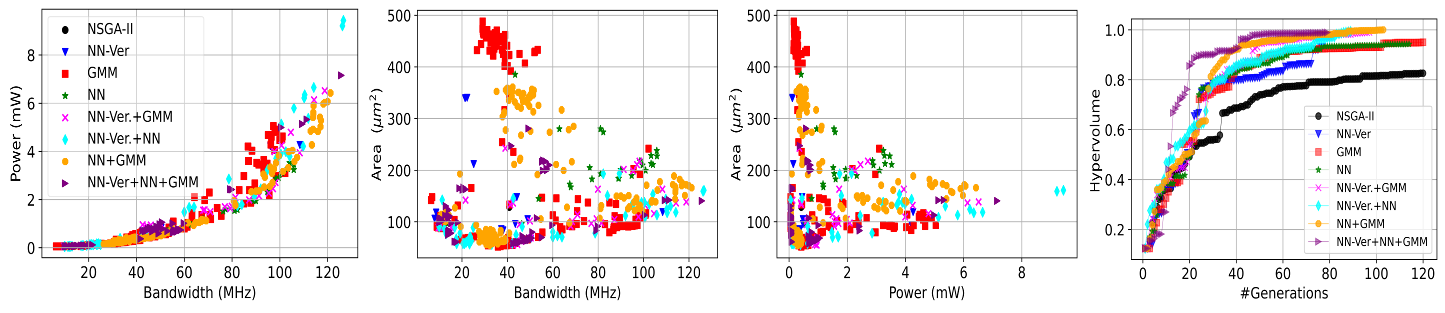
Fig. 10: Three-stage differential transimpedance amplifier topology [5].

Fig. 11(a)-(c) show the NDFs related to *power*, *bandwidth*, and *area* in Step 1. Note that some of the solutions have large area, as shown in Fig. 11(b). However, they have a better power or bandwidth and lie on the NDF. Fig. 11(d) shows the hypervolume obtained using the eight methods in Step 1. The number of simulations and time (sim.+alg.) needed for each synthesis methodology are as follows: (1) GA: 11,415 sim., 78.1+0.9 min; (2) NN-Ver.: 7,496 sim., 56.6+122.0 min; (3) GMM: 11,804 sim., 76+62.5 min; (4) NN: 11,189 sim., 69.9+95.3min; (5) NN-Ver.+GMM: 9,510 sim., 59.5+103.4 min; (6) NN-Ver.+NN: 8,696 sim.,54.5+115.1 min; (7) NN+GMM: 10,104 sim.,71.0+116.7 min; (8) NN-Ver.+NN+GMM: 7,778 sim., 55.2+145.9 min.

The top row of Fig. 12 depicts the Step 1 solution with a black circle. This is dominated using the inverse design methods in Step 2. The number of simulations and the time (sim.+alg.) required in Step 2 are as follows: (1) NN-Ver.: 6,667 sim., 2,446+10,697s; (2) GMM: 8,980 sim., 3,609+18,499s; (3) NN: 15,785 sim., 5,578+30,549s; (4) NN-Ver.+GMM: 9,563 sim., 3,716+32,773s; (5) NN-Ver.+NN: 14,910 sim., 5,430+30,737s; (6) NN+GMM: 6,650 sim., 2,409+9,338s; (7) NN-Ver.+NN+GMM: 11,024 sim., 4,130+32,322s. The bottom row of Fig. 12 shows the solution from ASSENT using a '+' sign. We observe that several inverse design solutions lie on the NDF. In the top part of Table IV, we compare the solution obtained using INFORM with the solutions using other methodologies. We observe that INFORM achieves a speedup of up to 29× (GMM first run in Step 2) for synthesis of a design comparable to ASSENT. NN-Ver.+NN+GMM generates a solution with 5.2% lower area and a similar power, gain,

TABLE III: Comparison of INFORM designs that dominate ASSENT [16] and human design [5] for the two-stage transimpedance amplifier. Parenthesis entries indicate the objective values when we first obtain a solution using INFORM that dominates the ASSENT or human [5] solution. Results shown for two different runs separated by a '/'. The hard constraint violation is encircled.

	#Samples	Time	Noise (pA/$\sqrt{\text{Hz}}$)	Gain (dB Ω)	Peaking (dB)	Power (mW)	Gate area (μm^2)	Bandwidth (GHz)
Spec	-		$\leq$ 19.3	$\geq$ 57.6	$\leq$ 1	min	min	max
Human Design [5]	1,289,618	months	18.6	57.7	0.927	8.11	23.11	5.95
DDPG [5]	50,000	30 GPU hrs	19.2	58.1	0.963	3.18	-	5.78
Bayesian Opt. [5]	880	30 hrs	$\boxed{19.6}$	58.6	0.629	4.24	-	5.16
ASSENT [16]	7,853	77 hrs	19.2	57.6	0.949	5.60	18.15	6.12
INFORM (Step 1 - GMM)	7,139	0.75+0.89 hrs	19.2	58.3	0.804	5.76	18.02	5.95
INFORM (Step 2 - NN-Ver.)	9,224/ 10,845	0.99+14.06/ 1.03+13.99hrs	19.2/ 19.2	57.8/ 57.8	0.944/ 0.948	5.34/ 5.36	17.44/ 16.95	6.08/ 6.08
INFORM (Step 2 - GMM)	13,572(7,922)/ 5,659(2,571)	1.42+13.63(0.84+4.76)/ 0.59+2.95(0.27+1.23) hrs	19.2(19.2)/ 19.2(19.2)	57.6(57.6)/ 57.6(57.6)	0.948(0.947)/ 0.947(0.941)	5.46(5.24)/ 5.47(5.50)	17.56(17.20)/ 17.42(17.53)	6.13(6.12)/ 6.13(6.13)
INFORM (Step 2 - NN)	14,004(11665)/ 14,324(12,015)	1.37+5.29(1.13+4.77)/ 1.49+5.69(1.25+5.10) hrs	19.2(19.2)/ 19.2(19.2)	57.6(57.6)/ 57.6(57.6)	0.948(0.948)/ 0.949(0.949)	5.55(5.55)/ 5.57(5.57)	17.5(17.5)/ 18.1(18.1)	6.12(6.12)/ 6.12(6.12)
INFORM (Step 2 - NN-Ver.+GMM)	9,186(6,629) 10,681(7,444)	1.02+14.0(0.73+10.52)/ 1.16+13.88(0.81+10.12) hrs	19.2(19.2)/ 19.2(19.2)	57.6(57.6)/ 57.6(57.7)	0.948(0.906)/ 0.947(0.949)	5.60(5.59)/ 5.50(5.46)	17.85(17.87)/ 17.57(17.55)	6.14(6.12) 6.13(6.12)
INFORM (Step 2 - NN-Ver.+NN)	14,690(11,504)/ 12,775(10,376)	1.56+13.47(1.23+10.86)/ 1.65+13.37(1.36+10.92) hrs	19.2(19.2)/ 19.2(19.2)	57.6(57.6)/ 57.6(57.6)	0.943(0.946)/ 0.947(0.940)	5.57(5.49)/ 5.47(5.46)	17.60(17.67)/ 17.57(17.52)	6.13(6.12)/ 6.13(6.12)
INFORM (Step 2 - NN+GMM)	7,982(5,455)/ 8,307(7,227)	0.90+9.13(0.61+6.61)/ 0.92+9.11(0.80+8.09) hrs	19.2(19.2)/ 19.2(19.2)	57.6(57.6)/ 57.6(57.6)	0.948(0.943)/ 0.944(0.944)	5.55(5.55) 5.53(5.53)	17.56(17.49)/ 17.73(17.73)	6.12(6.12)/ 6.12(6.12)
INFORM (Step 2 - NN-Ver.+NN+GMM)	9,532(3,843)/ 13,606(6,998)	1.0+8.5(0.40+2.79)/ 1.47+13.15(0.75+6.86) hrs	19.2(19.2)/ 19.2(19.2)	57.6(57.6)/ 57.6(57.6)	0.947(0.948)/ 0.946(0.943)	5.54(5.57)/ 5.54(5.47)	17.72(17.66)/ 17.52(17.38)	6.13(6.13)/ 6.13(6.12)
INFORM (dominate Human) (Step 2 - NN-Ver.)	7,648(140)/ 7,505(1,993)	0.86+9.20(0.02+0.02)/ 0.83+9.20(0.22+1.04) hrs	18.6(18.6)/ 18.6(18.6)	57.7(57.7)/ 57.7(57.7)	0.927(0.878)/ 0.925(0.926)	7.20(7.55)/ 7.74(7.94)	22.35(22.27)/ 23.03(22.92)	6.05(5.96)/ 6.03(5.95)
INFORM (dominate Human) (Step 2 - NN-Ver.+NN+GMM)	10,295(4,310)/ 10,172(3,069)	1.01+9.03(0.42+1.69)/ 1.01+9.03(0.30+0.86) hrs	18.6(18.6)/ 18.6(18.5)	57.7(57.7)/ 57.7(57.8)	0.924(0.901)/ 0.926(0.915)	7.50(7.86)/ 7.64(8.01)	22.93(23.04)/ 22.77(23.00)	6.04(5.97)/ 6.04(5.99)



(a) (b) (c) (d)

Fig. 11: Three-stage transimpedance amplifier optimization. NDFs for (a) power vs. bandwidth, (b) area vs. bandwidth, (c) area vs. power, and (d) comparison of hypervolume using the eight methods in Step 1.

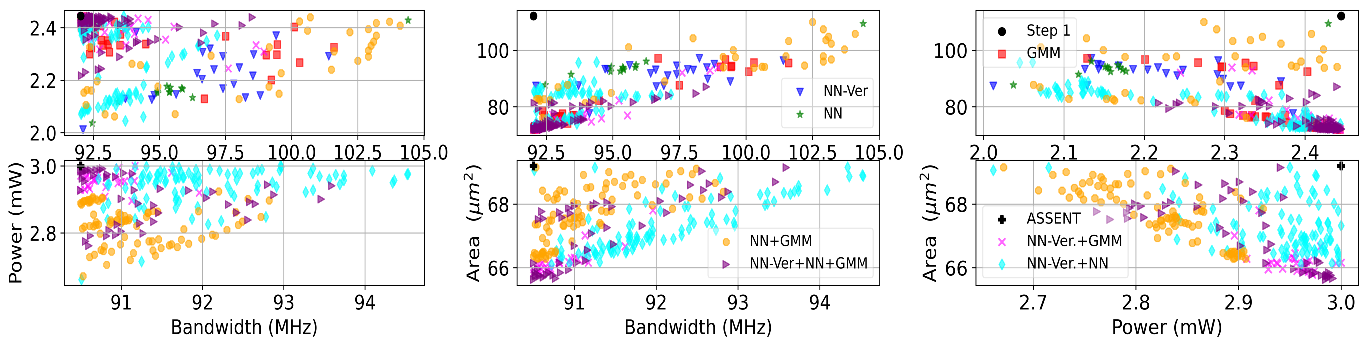
and bandwidth compared to the ASSENT solution, however, at a reduced speed-up of 6×. We also use Step 2 of INFORM to drive the Step 1 (human) solution to be better in all the performance metrics (objectives+constraints) than the DDPG (human) solution in [5]. When synthesizing the solution that dominates the human design, we change the search space to lie within ±50% of the human design. The solution obtained using INFORM has about 19% (33%) lower area, and similar performance in other metrics in comparison to the DDPG (human) design.

Next we use Step 2 to drive one solution to another solution. The bottom half of Table IV shows results when the first INFORM solution generated by GMM that dominates the ASSENT solution is driven to another solution. We aim to decrease (increase) power (bandwidth) while setting other constraints to match that of the ASSENT solution. We show the additional samples/time required with respect to the GMM solution obtained in Step 2. We observe that power (bandwidth) is 14.7% (5.3%) lower (higher) than the ASSENT solution.

VII. Discussion and Limitations

Next, we discuss the main highlights of INFORM and its limitations. A designer may be interested in fine-tuning an existing solution with the goal of either improving the value of the objective function or satisfying a different set of constraints. In such scenarios, the designer can search around the solution by simply using Step 2 as illustrated in the experiment section. INFORM generates multiple candidate solutions in parallel, thus allowing the use of multiple CPU cores for simulation. The designer can easily trade off sample efficiency with total synthesis time by increasing the number of candidate solutions generated using inverse design. It continuously learns to improve the value of the objective function, as opposed to using episodes that require frequent resets in RL. We only use the CPU for optimization, thus presenting a cheaper alternative to contemporary optimization techniques that rely on GPUs to solve an RL or a BO problem.

We use an NN to model the complex nonlinear system objectives and NN-Ver. for inverse design to help achieve sample efficiency. Note that any NN-Ver., e.g., Marabou [29], could also have been used instead of NSVerify [20]. GMM provides a distribution over the output space instead of just a point estimate as in an NN, thus providing a new dimension to inverse design. Finally, NN-based inverse design is a simple technique that maps the objectives/constraints to the component values. We also note that NN (in NN-based inverse design) can be replaced by any other optimizer such as Xgboost [30].



(a) (b) (c)

Fig. 12: Dominating the Step 1 (top) and ASSENT solution (bottom) shown in black for the three-stage transimpedance amplifier: (a) power vs. bandwidth, (b) area vs. bandwidth, and (c) area vs. power.

TABLE IV: Comparison of designs synthesized using INFORM (for two different runs separated by a '/') with those synthesized using ASSENT [16] and [5] for the three-stage transimpedance amplifier (top part). The bottom part shows results for the case when the first solution, which dominates the ASSENT solution, obtained using GMM is driven to another solution.

	#Samples	Time	Bandwidth (MHz)	Gain (dBΩ)	Power (mW)	Gate area (μm^2)
Spec	-	-	max	$\geq$ 80	min	min
Human Design [5]	10,000,000	months	90.1	80.09	1.37	211.0
DDPG [5]	40,000	40 GPU hrs	92.5	80.30	2.50	90.0
Bayesian Opt. [5]	1,160	40 hrs	72.5	80.47	4.25	130.0
ASSENT [16]	5,749	84.7 hrs	90.5	80.04	3.00	69.2
INFORM (Step 1 - NN-Ver.+NN)	8,696	0.91+1.92 hrs	92.02	84.33	2.45	112.13
INFORM (Step 2 - NN-Ver.)	5,730(1,735)/ 6,931(4,017)	0.59+9.91(0.18+4.48)/ 0.73+10.12(0.42+7.95) hrs	90.5(90.8)/ 90.5(90.5)	80.05(80.47)/ 80.05(80.05)	2.92(2.92)/ 2.67(2.67)	67.2(68.4)/ 69.0(69.2)
INFORM (Step 2 - GMM)	11,640(565)/ 6,147(2,159)	1.24+12.41(0.06+0.04)/ 0.71+2.52(0.25+0.46) hrs	90.6(91.3)/ 90.5(91.2)	80.05(80.28)/ 80.05(80.25)	2.96(2.87)/ 2.87(2.86)	66.6(66.8)/ 66.9(68.8)
INFORM (Step 2 - NN)	15,144(8,860)/ 20,391	1.50+9.04(0.88+5.05)/ 2.20+12.83 hrs	90.5(90.8)/ 90.5	80.05(80.07)/ 80.06	2.94(2.83) / 2.67	67.3(69.05) 70.0
INFORM (Step 2 - NN-Ver.+GMM)	9,604(1,678)/ 10,102(1,172)	1.13+9.35(0.19+0.70)/ 1.18+7.41(0.14+0.84) hrs	90.6(90.9)/ 90.5(90.5)	80.06(80.26)/ 80.05(80.21)	2.97(2.88)/ 2.98(2.96)	65.8(68.4)/ 65.9(68.8)
INFORM (Step 2 - NN-Ver.+NN)	20,600(8,848)/ 23,416(4,334)	2.11+12.94(0.91+6.43)/ 2.44+11.89(0.45+3.69) hrs	90.5(90.5)/ 90.7(92.4)	80.06(80.06)/ 80.07(80.17)	2.99(2.66)/ 2.98(2.98)	65.8(69.1)/ 66.4(68.9)
INFORM (Step 2 - NN+GMM)	11,361(959)/ 8,400(2,162)	1.15+5.96(0.10+0.22)/ 0.86+3.62(0.22+0.56) hrs	90.5(91.3)/ 90.6(91.0)	80.06(80.16)/ 80.05(80.05)	2.90(2.80)/ 2.89(2.85)	66.3(69.0)/ 66.9(68.6)
INFORM (Step 2 - NN-Ver.+NN+GMM)	15,365(1,757)/ 16,364(2,646)	1.74+9.89(0.20+0.54)/ 1.85+11.30(0.31+1.71) hrs	90.5(91.4)/ 90.5(90.6)	80.05(80.26)/ 80.06(80.24)	2.99(2.94)/ 2.97(2.83)	65.6(68.9)/ 65.6(68.8)
INFORM (dominate DDPG) (Step 2- NN-Ver.+NN+GMM)	12,916(450)/ 11,280(407)	1.48+9.65(0.05+0.07)/ 1.17+6.80(0.04+0.05) hrs	92.5(95.3)/ 92.5(93.1)	80.32(80.93)/ 80.33(80.34)	2.50(2.37)/ 2.50(2.10)	72.7(87.4)/ 72.7(89.3)
INFORM (dominate Human) (Step 2- NN-Ver.+NN+GMM)	13,493(1,319)/ 13,212(689)	1.53+13.62(0.15+0.34)/ 1.41+13.64(0.07+0.23) hrs	90.6(90.1)/ 90.3(90.3)	80.16(80.09)/ 80.10(80.17)	1.36(1.37)/ 1.36(1.37)	141.7(211.0)/ 144.6(208.5)
INFORM (Bandwidth max.) (NN-Ver.+NN+GMM)	17,563/ 7,027	1.93+13.11/ 0.82+4.12 hrs	95.3/ 94.7	80.05/ 80.06	3.00/ 2.98	69.2/ 69.2
INFORM (Power min.) (NN-Ver.+NN+GMM)	7,763/ 5263	0.90+4.61/ 0.61+2.67 hrs	90.5/ 90.6	80.05/ 80.06	2.56/ 2.57	69.2/ 69.2

A human expert in the loop of the design process can further enhance the process. In Step 1, human knowledge can enable focus only on the relevant range of the objective values. After running Step 1 for the three-stage transimpedance amplifier, we can see that many designs have a very low area and power that are perhaps uninteresting. We observe similar phenomena in the second electrical circuit example too.

INFORM is limited to optimization of a fixed system architecture. Recent techniques such as GCN-RL [31] use transfer learning across different electrical circuit topologies and a weighted sum of different outputs during optimization. INFORM can be extended similarly by using simulation data from one system architecture to optimize another system architecture to solve a CMOO problem.

VIII. Conclusion

This article proposed a framework called INFORM to solve real-world CMOO problems. We enhanced a GA by injecting candidate solutions using seven inverse design methods. The inverse design method generates multiple candidate solutions simultaneously, thus speeding up synthesis. We made the design process sample-efficient by performing targeted simulations in Step 2. Using INFORM, we demonstrated reduction in synthesis time by 29× relative to ASSENT, a state-of-the-art system synthesis method, while delivering similar or better performance (up to 33%) compared to a human expert.

Acknowledgment: We would like to thank Kenza Hamidouche, Hossein Valavi, Jaime Fernández Fisac, Brendon Englot, Sanjai Narain, Kishore Pochiraju, Dana Chee, Sean Rooney, and Emil J. Pitz for guidance and help with the setup for several benchmarks used for evaluating INFORM. We also thank Shrikanth Narayanan, Prasanta K. Ghosh, and the Indo-US Science and Technology Forum - Viterbi program for providing an internship at the University of Southern California, from where we got the inspiration for GMM-based inverse design. The simulations presented in this article were performed on computational resources managed and supported by Princeton Research Computing at Princeton University.

References

[1] D. A. Van Veldhuizen and G. B. Lamont, "Evolutionary computation and convergence to a Pareto front," in *Proc. Late-breaking Papers at the Genetic Programming Conference*, 1998, pp. 221–228.

[2] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[3] M. T. Emmerich and A. H. Deutz, "A tutorial on multiobjective optimization: Fundamentals and evolutionary methods," *Natural Computing*, vol. 17, no. 3, pp. 585–609, 2018.

[4] S. Daulton, M. Balandat, and E. Bakshy, "Differentiable expected hypervolume improvement for parallel multi-objective Bayesian optimization," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33, 2020, pp. 9851–9864.

[5] H. Wang, J. Yang, H.-S. Lee, and S. Han, "Learning to design circuits," in *NeurIPS Machine Learning for Systems Workshop*, 2018.

[6] K. Settaluri, A. Haj-Ali, Q. Huang, K. Hakhamaneshi, and B. Nikolic, "AutoCkt: Deep reinforcement learning of analog circuit designs," in *Proc. Design, Automation & Test in Europe Conference & Exhibition*, 2020, pp. 1–6.

[7] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi *et al.*, "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 7862, pp. 207–212, 2021.

[8] G. Zhang, H. He, and D. Katabi, "Circuit-GNN: Graph neural networks for distributed circuit design," in *Proc. Int. Conf. on Machine Learning*, ser. Proc. of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 7364–7373.

[9] W. Lyu, F. Yang, C. Yan, D. Zhou, and X. Zeng, "Batch Bayesian optimization via multi-objective acquisition ensemble for automated analog circuit design," in *Proc. Int. Conf. Machine Learning*, 2018, pp. 3306–3314.

[10] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, Part I: Solving problems with box constraints," *IEEE Trans. on Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, 2014.

[11] W. Lyu, F. Yang, C. Yan, D. Zhou, and X. Zeng, "Multi-objective Bayesian optimization for analog/RF circuit synthesis," in *Proc. Annual Design Automation Conf.*, 2018, pp. 1–6.

[12] Z. Gao, J. Tao, F. Yang, Y. Su, D. Zhou, and X. Zeng, "Efficient performance trade-off modeling for analog circuit based on Bayesian neural network," in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, 2019, pp. 1–8.

[13] B. De Smedt and G. Gielen, "WATSON: Design space boundary exploration and model generation for analog and RFIC design," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 2, pp. 213–224, 2003.

[14] M. Chu and D. Allstot, "Elitist nondominated sorting genetic algorithm based RF IC optimizer," *IEEE Trans. on Circuits and Systems I*, vol. 52, no. 3, pp. 535–545, 2005.

[15] S. Narain, E. Mak, D. Chee, B. Englot, K. Pochiraju, N. K. Jha, and K. Narayan, "Fast design space exploration of nonlinear systems: Part I," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 9, pp. 2970–2983, 2022.

[16] P. Terway, K. Hamidouche, and N. K. Jha, "Fast design space exploration of nonlinear systems: Part II," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 9, pp. 2984–2999, 2022.

[17] M. Balandat, B. Karrer, D. R. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy, "BoTorch: A framework for efficient Monte-Carlo Bayesian optimization," in *Advances in Neural Information Processing Systems, vol. 33*, 2020.

[18] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," 2016. [Online]. Available: https://gym.openai.com

[19] C. Qin and M. A. Carreira-Perpiñán, "Estimating missing data sequences in X-ray microbeam recordings," in *Proc. Eleventh Annual Conf. Int. Speech Communication Association*, 2010.

[20] M. Akintunde, A. Lomuscio, L. Maganti, and E. Pirovano, "Reachability analysis for neural agent-environment systems," in *Proc. Int. Conf. Principles of Knowledge Representation and Reasoning*, 2018.

[21] M. D. McKay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 42, no. 1, pp. 55–61, 2000.

[22] S. Burhenne, D. Jacob, and G. P. Henze, "Sampling based on Sobol sequences for Monte Carlo techniques applied to building simulations," in *Proc. Int. Conf. Build. Simulation*, 2011, pp. 1816–1823.

[23] S. I. Vrieze, "Model selection and psychological theory: A discussion of the differences between the Akaike information criterion (AIC) and the Bayesian information criterion (BIC)," *Psychological Methods*, vol. 17, no. 2, p. 228, 2012.

[24] F. Chollet *et al.* (2015) Keras. [Online]. Available: https://github.com/fchollet/keras

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[26] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2020. [Online]. Available: http://www.gurobi.com

[27] D. Izzo, "PyGMO and PyKEP: Open source tools for massively parallel optimization in astrodynamics (the case of interplanetary trajectory optimization)," in *Proc. Int. Conf. on Astrodynamics Tools and Techniques*, 2012.

[28] J. Blank and K. Deb, "PyMOO: Multi-objective optimization in Python," *IEEE Access*, vol. 8, pp. 89 497–89 509, 2020.

[29] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić *et al.*, "The Marabou framework for verification and analysis of deep neural networks," in *Proc. Int. Conf. on Computer-Aided Verification*, 2019, pp. 443–452.

[30] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, 2016, pp. 785–794.

[31] H. Wang, K. Wang, J. Yang, L. Shen, N. Sun, H.-S. Lee, and S. Han, "GCN-RL circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning," in *Proc. ACM/IEEE Design Automation Conference*, 2020, pp. 1–6.

Prerit Terway received his M.S. degree from University of Michigan, Ann Arbor, and B.Tech. degree from Indian Institute of Technology, Gandhinagar, India, both in Electrical Engineering. He is currently a Ph.D. candidate in Electrical and Computer Engineering at Princeton University. His research interests include machine learning, active learning, and cyber-physical systems.

Niraj K. Jha received his B.Tech. degree in Electronics and Electrical Communication Engineering from Indian Institute of Technology, Kharagpur, India in 1981 and Ph.D. degree in Electrical Engineering from University of Illinois at Urbana-Champaign, IL in 1985. He has been a faculty member of the Department of Electrical and Computer Engineering, Princeton University, since 1987. He is a Fellow of IEEE and ACM, and was given the Distinguished Alumnus Award by I.I.T., Kharagpur, in 2014. He has also received the Princeton Graduate Mentoring Award. He has served as the Editor-in-Chief of IEEE Transactions on VLSI Systems and as an Associate Editor of several other journals. He has co-authored five books that are widely used. His research has won 20 best paper awards or nominations. He has published more than 460 papers and received 23 patents. His research interests include smart healthcare, cybersecurity, and machine learning. He has given several keynote speeches in the areas of nanoelectronic design/test, smart healthcare, and cybersecurity.