"Smart" Design Space Sampling to Predict Pareto-Optimal Solutions

Marcela Zuluaga
Department of Computer Science
ETH Zurich
zuluaga@inf.ethz.ch

Andreas Krause
Department of Computer Science
ETH Zurich
krausea@ethz.ch

Peter Milder
Department of Electrical and Computer Engineering
Carnegie Mellon University
pam@ece.cmu.edu

Markus Püschel
Department of Computer Science
ETH Zurich
pueschel@inf.ethz.ch

Abstract

Many high-level synthesis tools offer degrees of freedom in mapping high-level specifications to Register-Transfer Level descriptions. These choices do not affect the functional behavior but span a design space of different cost-performance tradeoffs. In this paper we present a novel machine learning-based approach that efficiently determines the Pareto-optimal designs while only sampling and synthesizing a fraction of the design space. The approach combines three key components: (1) A regression model based on Gaussian processes to predict area and throughput based on synthesis training data. (2) A "smart" sampling strategy, GP-PUCB, to iteratively refine the model by carefully selecting the next design to synthesize to maximize progress. (3) A stopping criterion based on assessing the accuracy of the model without access to complete synthesis data. We demonstrate the effectiveness of our approach using IP generators for discrete Fourier transforms and sorting networks. However, our algorithm is not specific to this application and can be applied to a wide range of Pareto front prediction problems.

Categories and Subject Descriptors B.5.2 [*Hardware*]: Design Aids—Automatic synthesis; F.2.2 [*Mathematics of Computing*]: Probability and Statistics—Probabilistic algorithms; I.5.1 [*Computing Methodologies*]: Pattern Recognition—Models

General Terms Algorithms, Performance

Keywords Pareto Optimality, High-Level Synthesis, Area and Performance Estimation, Machine Learning

1. Introduction

The ultimate goal of high-level synthesis tools is to free designers from dealing with implementation details that do not affect behavioral specifications. Users only specify the required functionality, and a Register-Transfer Level (RTL) description is automatically generated. One example of such a tool is an Intellectual Property (IP) generator that emits RTL designs for a particular function or operation. However, there are usually important degrees of freedom in the process of generating designs from a high-level description. These do not affect the functional behavior but may represent different cost-performance tradeoffs. One example of such a tradeoff is area cost versus throughput performance. The designer is only interested in those designs that are Pareto-optimal to select the best solution for the constraints at hand. Obtaining these designs, however, is costly since it requires the synthesis and evaluation of all degrees of freedom supported by the tool.

As an example, consider an IP generator for Sorting Networks developed in prior work [19]. The generator can produce RTL descriptions for a given input size n (the length of the list to be sorted), and a set of configuration parameters c that affect area and

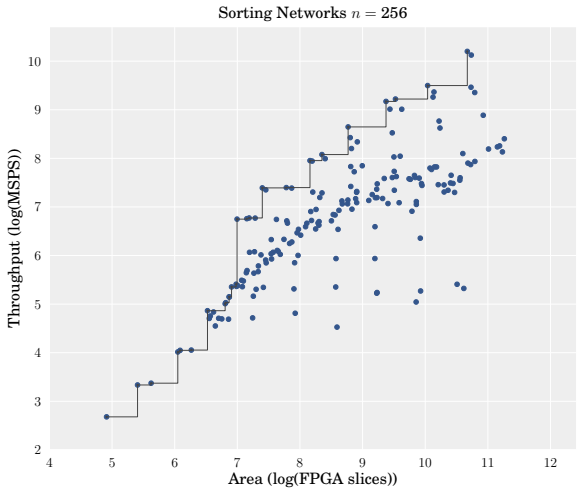

Figure 1. Design space of sorting networks, supported by an IP generator, for input size 256.

LCTES 2012, June 12-13, 2012, Beijing, China

throughput. For $n = 256$, the generator offers 207 configurations c; synthesis of each yields the area/throughput plot in Fig. 1. The Pareto-optimal designs are joined by a line and represent only a fraction of the design space. The question raised is how to find the Pareto-optimal designs efficiently. An ideal solution is a model that avoids actual synthesis. However, building by hand such a model that covers all cases, even in the case of a fixed-function generator, is notoriously difficult, not least because the parameters in c are usually heterogeneous and have complex interactions and effects on the final results. Thus, an attractive solution is to generate such a model from synthesis data using techniques from machine learning. This is the approach taken in this paper.

Contribution. In this paper we address the problem of finding the Pareto-optimal designs efficiently, i.e., with minimal sampling and synthesis of the design space. Our approach uses a novel machine learning algorithm that is specifically designed for the required multi-objective optimization. The algorithm iteratively and "smartly" samples the design space to build a continuously improving model until a stopping criterion is met. It consists of three main components: (1) A model for predicting area and throughput based on Gaussian process (GP) regression. This approach proves to be well-suited to deal with the heterogeneous configuration space and the non-linear objective functions. As a major advantage, the model captures predictive uncertainty, which is crucial for guiding our sampling and termination strategy, as explained next. (2) A novel sampling strategy, referred to as GP-PUCB, to iteratively refine the above model. The strategy is designed to both maximally improve prediction accuracy while focusing on refining the model in the Pareto region, i.e., the region of interest. Our approach adapts the known upper confidence bound rule for single objective Gaussian processes [13] to our multi-objective case. (3) A stopping criterion that can assess the accuracy of the model even though synthesis results are (of course) not available for the entire design space.

We evaluate our algorithm with two IP generators, one for sorting networks [19] and one for discrete Fourier transforms [8]. As a metric for comparing actual versus estimated Pareto front we use the logarithmic hypervolume [16], which proportionally penalizes mispredictions across the objective space. The results show that sampling a small fraction of the design space can indeed yield predictions of the Pareto front that are sufficiently accurate to be of use in applications.

Finally, we would like to emphasize that our algorithm is not specific to the IP generators or scenario we use in our empirical case study. Due to the generality of the problem attacked, we expect it to be a viable choice for a wide range of Pareto front prediction problems.

2. Background on our Case Study

In this paper we consider two particular IP generators as case studies for the approach and evaluation. Both can generate hardware for a variety of input sizes n, and for each n there are various configurations c that determine different tradeoffs between area and throughput. The exact working of the generators is not of importance here, only the parameterization of the design space.

Discrete Fourier transform (DFT). The IP generator in [8] generates RTL Verilog for DFT, given the input size n, and a configuration with four parameters: $c = (\text{algorithm, radix, depth, streaming width})$. The first specifies the fast Fourier transform algorithm, the second the associated radix. The depth ℓ and the streaming width w are implementation choices: ℓ indicates the number of pipelined computation stages, and w the number of samples that are input at every clock cycle. The generated core includes $w\ell/2$ basic butterfly units.

Sorting network (SNW). Similar to [8], the SNW generator in [19] generates RTL Verilog for sorters, given the length n of the sequence to be sorted and a configuration with 3 parameters: $c = (\text{algorithm, depth, streaming width})$. The first specifies the variant of bitonic sorter to be used, the others are as before. The generated core has $w\ell/2$ size-2 sorters.

3. Problem Formulation

We wish to find for given input size n the set of Pareto-optimal designs while testing as few configurations as possible. Notice that while we focus on a dual-criteria optimization—trading off area and throughput—the approach can be easily generalized to multi-criteria optimization with an arbitrary number of objective functions [3]. We now introduce notation to provide a formal problem statement and to emphasize the general nature of the setting we consider. This should make our approach suitable for a much wider range of applications than studied here.

Our target IP generators produce designs that are fully specified by the input size n and the configuration vector c. The set of "admissible" choices for c depends on n. We denote with D_n the design space of configurations[1] available for a fixed n. Formally,

$$D_n = \{d = (n, c) \mid c \text{ admissible}\}.$$

If N is the finite set of all supported sizes n, then

$$D = \bigcup_{n \in N} D_n$$

is the entire design space supported by the generator.

We consider two objective functions, $\mathcal{A}$ (area) and $\mathcal{T}$ (throughput) that map every $d \in D$ to $(\mathcal{A}(d), \mathcal{T}(d))$ in the objective space (pairs of positive reals). Evaluating $\mathcal{A}$ and $\mathcal{T}$ involves actual synthesis and is thus expensive. Further, we define the Pareto dominance relation in D_n: $d \succeq d'$ if $\mathcal{A}(d) \leq \mathcal{A}(d')$ and $\mathcal{T}(d) \geq \mathcal{T}(d')$. The set $P_n \subseteq D_n$ of Pareto-optimal designs is thus the set of maximal points in this order.

Our goal is to predict $P_n, n \in N$, as accurately as possible from the evaluation (synthesis) of only a small subset $S \subset D$. In order to do that, we need to predict area and throughput for designs that are not synthesized. In the following, we describe how we can use machine learning to make such predictions.

4. Generating Predictions

Our approach to predicting the set of Pareto optimal configurations P_n for a given input size n is to learn models of the functions $\mathcal{A}$ and $\mathcal{T}$. The models will be denoted with $\hat{\mathcal{A}}$ and $\hat{\mathcal{T}}$, respectively. Obtaining a prediction $\hat{P}_n$ of P_n for some n, is thus straightforward through cheap evaluation of $\hat{\mathcal{A}}$ and $\hat{\mathcal{T}}$ on all $d \in D_n$ and identifying designs considered Pareto-optimal.

Generalizing across sizes. Since designs for different sizes n share patterns, we build models for the entire design space D instead of for the D_n separately. Since $\mathcal{A}$ and $\mathcal{T}$ tend to grow with n, we normalize these functions with suitable scaling factors determined by d (i.e., no synthesis is needed) to aid comparison across n. We refer to our normalized functions as $\mathcal{A}'$ and $\mathcal{T}'$:

$$\begin{aligned} \mathcal{A}'(d) &= \mathcal{A}(d)/\beta(d), \\ \mathcal{T}'(d) &= \mathcal{T}(d)/\gamma(d). \end{aligned} \tag{1}$$

In our case we choose $\beta(d) = w\ell/2$ (number of basic elements) and for $\gamma(d)$ the average number of input samples per cycles. For example, for fully streaming designs, $\gamma(d) = w$.

In summary, we learn models (or predictors) $\hat{\mathcal{A}}'$ and $\hat{\mathcal{T}}'$. The models $\hat{\mathcal{A}}$ and $\hat{\mathcal{T}}$ are obtained by simple denormalizing.

Training stage. Alg. 1 shows how, given the design space D, we generate the predictors $\mathcal{A}'$ and $\mathcal{T}'$ from a growing set

[1] We will use "design" and "configuration" interchangeably.

Algorithm 1: `train`. Training stage that learns area and throughput models. Input: D. Output: $\hat{\mathcal{A}}', \hat{\mathcal{T}}'$.

$S = \emptyset$ {S stores training data: $(s, \mathcal{A}'(s), \mathcal{T}'(s))$}
repeat
  if $|S| < 0.01 \cdot |D|$ **then**
    s = `random`$(D - S)$ {pick random element from $D - S$}
  else
    s = `smartSampling`$(D, S, \hat{\mathcal{A}}', \hat{\mathcal{T}}')$ {see Alg. 3}
  end if
  $(\mathcal{A}'(s), \mathcal{T}'(s))$ = `measure`(s) {run synthesis}
  $S = S \cup (s, \mathcal{A}'(s), \mathcal{T}'(s))$
  $(\hat{\mathcal{A}}', \hat{\mathcal{T}}')$ = `learn`(S) {see Section 4.1 }
until `stop`$(D, S, \hat{\mathcal{A}}', \hat{\mathcal{T}}')$ {see Alg. 6}
return $(\hat{\mathcal{A}}', \hat{\mathcal{T}}')$

Algorithm 2: `predict`. Predicting Pareto-optimal designs. Input: n, $\hat{\mathcal{A}}'$, $\hat{\mathcal{T}}'$. Output: $\hat{P}_n$.

$(Q_1, Q_2) =$ `predictPareto`$(D_n, \hat{\mathcal{A}}'(d), \hat{\mathcal{T}}'(d))$ {see Alg. 5}
$\hat{P}_n = Q_1 \cup Q_2$ {$\hat{P}_n \subset D_n$ contains all predicted Pareto optimal configurations}
return $\hat{P}_n$

of evaluated training data S. First, a random 1% of the design space is measured to create initial models. After that, samples are selectively chosen to maximize progress (explained in Section 4.2). In every iteration, a regression model based on Gaussian processes is trained and updated based on the new sample (explained in Section 4.1). Then a stopping criterion is evaluated to assess the maturity of the models (explained in Section 4.4).

Once the stopping criterion is met, the training terminates and the models are available for prediction.

Prediction stage. Alg. 2 shows how the models $\hat{\mathcal{A}}$ and $\hat{\mathcal{T}}$ are used to predict Pareto-optimal designs. The algorithm takes the uncertainty in predictions into account, as explained in Section 4.3.

4.1 Gaussian Process Modeling

We now describe the function `learn` in Alg. 1. Given a set of training data $(s, \mathcal{A}'(s), \mathcal{T}'(s))$, with $s \in S \subset D$, we create two independent regression models $\hat{\mathcal{A}}'$ and $\hat{\mathcal{T}}'$. While, in principle, many techniques could be used, we choose Gaussian Processes (GPs) [11] for reasons detailed below. In our implementation, we use the Gaussian Process Regression and Classification Toolbox for Matlab [10].

A GP is a statistical model that can capture complex, non-linear relationships between the configuration d and the objective functions $\mathcal{T}'(d)$ and $\mathcal{A}'(d)$. It is *nonparametric*, and therefore not limited to a particular parametric form (such as linearity). A GP is fully specified by a mean function and a covariance function. As a Bayesian model, a GP prior allows expression of the assumption that design points that are close in the space are likely to have similar values of the functions $\mathcal{T}'$ and $\mathcal{A}'$, and this similarity is captured by the covariance function[2]. Upon observation of the training set S, Bayesian inference is used to compute a posterior distribution over the $\hat{\mathcal{A}}'(d)$ and $\hat{\mathcal{T}}'(d)$. A crucial advantage of GPs is that the posterior provides closed-form predictive distributions for $\hat{\mathcal{A}}'(d)$ and $\hat{\mathcal{T}}'(d)$ with both a mean and a variance for designs d not in the training set S. The mean is the predicted value and the

[2] For our model we chose a constant mean function, and a composite covariance function equal to the sum of a linear covariance function and a squared exponential covariance function as explained in [11, pp. 83].

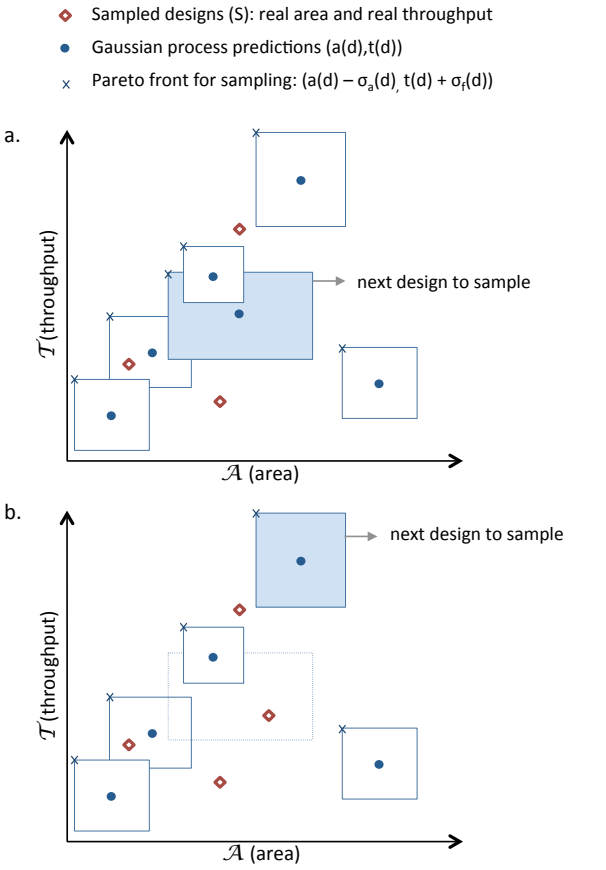

Figure 2. Sampling strategy

Algorithm 3: `smartSampling`. Choose a sample point in the design space D. Input: $D, S, \hat{\mathcal{A}}', \hat{\mathcal{T}}'$. Output: s.

$P_u = \emptyset$
$U_n = \emptyset$
for all $n \in N$ **do**
  for all $d \in D_n$ **do**
    $U_n = U_n \cup (d, (a(d) - \sigma_a(d), t(d) + \sigma_t(d))$ {values taken from $\hat{\mathcal{A}}(d)$ and $\hat{\mathcal{T}}(d)$ to find upper left corners of confidence rectangles}
  end for
  $P_u = P_u \cup$ `extractPareto`(U_n) {see Alg. 4}
end for
$s = p \in P_u$ with maximum $\sigma_a(p) + \sigma_t(p)$
return s

variance specifies the uncertainty in the prediction. For simplicity of notation, we denote with $a(d) = \hat{\mathcal{A}}(d)$ the prediction, and with $\sigma_a(d)$ the associated standard deviation provided by the model. Similarly, we define $t(d) = \hat{\mathcal{T}}(d)$ and $\sigma_t(d)$.[3] The covariance function and the mean function include parameters that control the smoothness of the functions estimated by the GP. These parameters are called *hyperparameters* and are learnt given the initial training samples S, and used in the GP posterior distribution in order to generate predictions over unseen design points.

[3] The standard deviation for $\hat{\mathcal{A}}(d)$ is obtained from the standard deviation of $\hat{\mathcal{A}}'(d)$ using the scaling factors in (1).

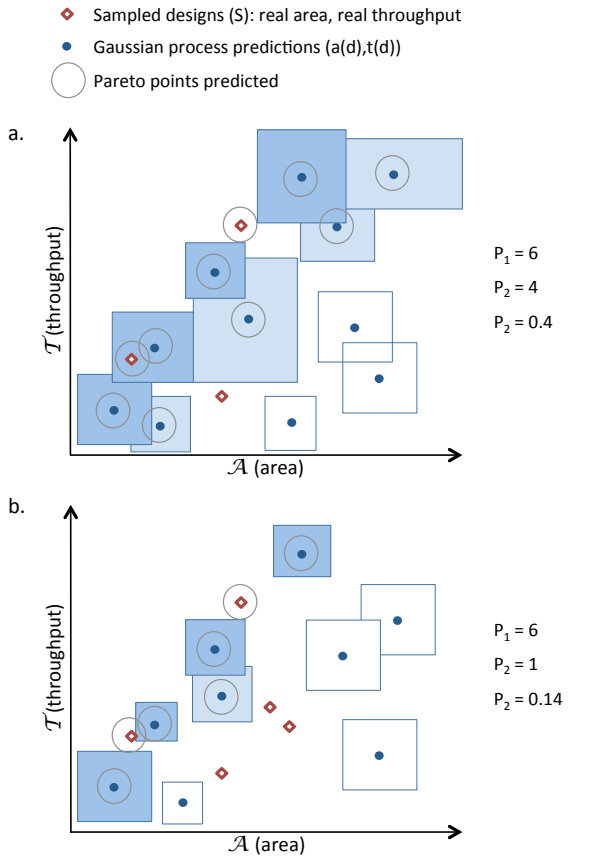

Figure 3. Stopping criterion

Algorithm 4: `extractPareto`. Extract Pareto optimal configurations. Input: E. Output: P.

$P_n = \emptyset$
for all $(d, \mathcal{A}(d), \mathcal{T}(d)) \in E$ **do**
  if $\nexists (d^*, \mathcal{A}(d^*), \mathcal{T}(d^*)) \in E$ with $d^* \preceq d$ and $d \not\preceq d^*$ **then**
    $P = P \cup d$ {E stores the vectors $(d, \mathcal{A}(d), \mathcal{T}(d))$}
  end if
end for
return P

4.2 Smart Sampling

Sampling the design space in a careful way is important as our goal is to minimize the number of measurements (syntheses) in the training stage. We propose GP-PUCB, a novel approach (the function `smartSampling` in Alg. 1) for selecting the next sample, which aims to reduce the uncertainty of the model and at the same time to focus on the predicted Pareto region of every D_n. To achieve this, we build on the Gaussian Process Upper Confidence Bound (GP-UCB) algorithm [13], which balances exploration of the space and emphasis on potential optimal designs (albeit only for single objective optimization).[4]

Fig. 2 illustrates our sampling strategy. After measuring a few sample designs, we are in the situation shown in part a. Here, three designs have been synthesized and measured; the results are

[4] The method considers a single objective maximization problem of a function $\mathcal{F}(x)$. A point x is chosen for sampling if it maximizes $\mu(x)+\eta\cdot\sigma(x)$ obtained from the current prediction; η is a chosen parameter.

Algorithm 5: `predictPareto`. Predict Pareto optimal configurations in D_n. Input: $D_n, \hat{\mathcal{A}}', \hat{\mathcal{T}}'$. Output: (Q_1, Q_2) such that $\hat{P}_n = Q_1 \cup Q_2$.

$E = \emptyset$ {E stores predicted values and their respective configuration vector d}
for all $d \in D_n$ **do**
  $E = E \cup (d, (a(d), t(d)))$
end for
Q_1 = `extractPareto`(E) {see Alg. 4}
$E = \emptyset$ {E now stores modified values of $\mathcal{A}$ and $\mathcal{T}$ their respective configuration vector d}
for all $d \in D_n$ and $\in Q_1$ **do**
  if $d \in Q_1$ **then**
    $E = E \cup (d, (a(d) + \sigma_a(d), t(d) - \sigma_t(d)))$
  else
    $E = E \cup (d, (a(d) - \sigma_a(d), t(d) + \sigma_t(d)))$
  end if
end for
Q_2 = `extractPareto`(E)
return (Q_1, Q_2)

Algorithm 6: `stop`. Obtain stopping condition to terminate training stage. Inputs: $D, \hat{\mathcal{A}}', \hat{\mathcal{T}}'$. Output: *stop*.

(Q_1, Q_2) = `predictPareto`$(D_n, \hat{\mathcal{A}}', \hat{\mathcal{T}}')$ {see Alg. 5}
$q_2 = \frac{|Q_2|}{|Q_1|+|Q_2|}$ {ratio of elements in Q_2}
if $q_2 < h$ **then**
  stop = True
else
  stop = False
end if{h is a threshold parameter}
return *stop*

red diamonds and have no uncertainty. For the other designs d, the current model offers predictions $(a(d), t(d))$ (blue dots) with uncertainty $a(d) \pm \eta \cdot \sigma_a(d)$ and $t(d) \pm \eta \cdot \sigma_t(d)$. We used $\eta = 1$, thus, uncertainties are represented as rectangles with width $2{\cdot}\sigma_a(d)$ and height $2 \cdot \sigma_t(d)$. The question is which design, i.e., rectangle, to sample and synthesize next.

We want to target Pareto optimal candidates, but also want to reduce the uncertainty (size of rectangle) of the model. To achieve both we optimistically consider all rectangles whose upper left (best) corner is Pareto-optimal; among those we pick the one with the largest uncertainty $\sigma_a(d)+\sigma_t(d)$. After measuring this sample, we are in the situation shown in Fig. 2, part b, and the process is repeated. The algorithm is formally described in Alg. 3.

4.3 Predicting Pareto-optimal Points

During training (Alg. 1), we obtain models $\hat{\mathcal{A}}'$ and $\hat{\mathcal{T}}'$ and hence the models $\hat{\mathcal{A}}$ and $\hat{\mathcal{T}}$, which for each design d provide a two-dimensional confidence region (rectangle), whose corner points are $(a(d) \pm \sigma_a(d),\ t(d) \pm \sigma_t(d))$. For the stopping criterion, we need to extract the Pareto-optimal points predicted so far.

The set of predicted Pareto-optimal points $\hat{P}_n$ is composed of two subsets Q_1 and Q_2. Q_1 is formed by the points that are not dominated given the mean values predicted by the model $(a(d), t(d))$. As a small misprediction in $(a(d), t(d))$ might change the dominance relationship among design points, we also predict points that are likely to be Pareto-optimal even though they are not given their predicted values $(a(d), t(d))$. Thus, Q_2 is composed of points that are not in Q_1 (since those are already predicted to be Pareto-optimal), and that are not dominated when points in Q_1 take values at the bottom-right corners (worst outcomes) in the confi-

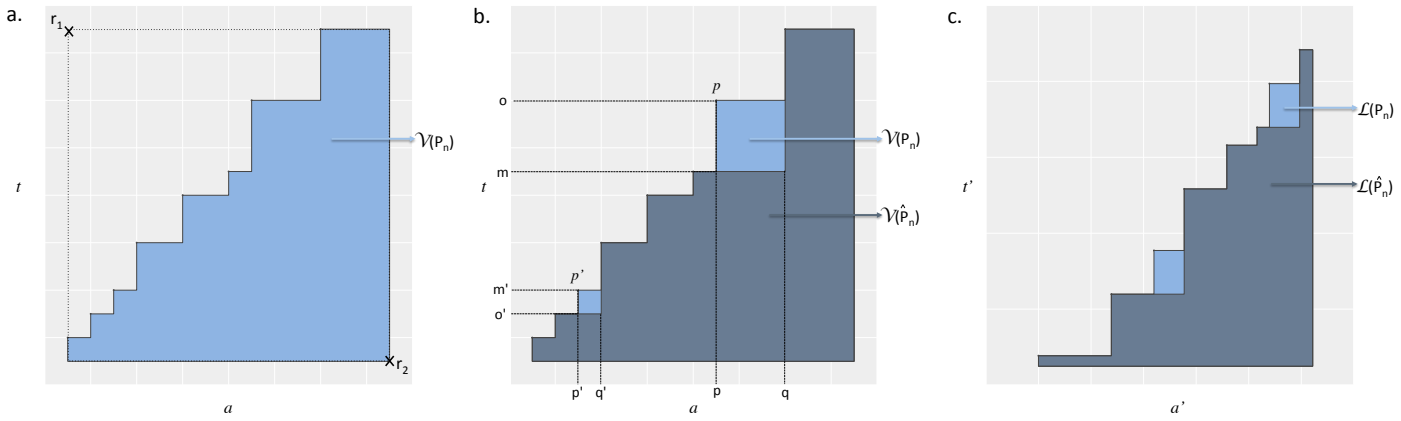

Figure 4. Examples of the hypervolume indicator ($\mathcal{V}$) and the logarithmic hypervolume indicator ($\mathcal{L}$), evaluated on the true Pareto optimal set P_n and the predicted Pareto optimal set $\hat{P}_n$.

dence region and points not in Q_1 take values at the top-left corner (best outcome). In Fig. 3(a), Q_2 consists precisely of the center points of the light blue rectangles. This is a conservative approach when confidence regions are still large. However, Q_2 tends to get smaller as more samples are taken and uncertainties are reduced.

The overall algorithm is formally described in Alg. 5 and is the crucial subroutine in the stopping criterion (Alg. 6) described next and also in Alg. 2.

4.4 Stopping Criterion

A stopping condition is used to terminate the training stage (`stop` in Alg. 1). When this condition is met, no more samples are evaluated and the model is used for predicting Pareto configurations as shown in Alg. 2. Our stopping criterion assesses the confidence of the current model along the Pareto front. The challenge is that without complete synthesis results, the accuracy cannot be assessed directly; hence, another approach is needed.

Using the notation from the previous section, for a given n, we define $q_2 = |Q_2|/|\hat{P}_n|$ as the ratio of points of designs that have been predicted as Pareto-optimal only due to uncertainty. Intuitively, as the model matures, the uncertainties are reduced and q_2 decreases. Once it drops below a chosen threshold, we terminate the training.

As example, Fig. 3(a) shows a stage at which the model is still immature and this is reflected by a large q_2. In Fig. 3(b) the model has become more mature, which is reflected by a smaller q_2.

The procedure is formalized in Alg. 6. We test several threshold values in Section 6.

5. Quality Indicator

The goal of our machine learning algorithm is to find the best approximation $\hat{P}_n$ of the Pareto set P_n for every input size n. To assess the approximation and compare different models and different stages of the training process, we hence need a measure of quality. As explained next, we use the logarithmic hypervolume, which is a special case of the weighted hypervolume suggested in [16].

5.1 Hypervolume

The *Hypervolume* indicator is a well known quality indicator for multi-objective optimization problems [17]. It measures the size of the space covered between the Pareto front and a reference point located inside the dominated region of the design space. Fig. 4(a) shows an example where the area that defines the hypervolume is highlighted in blue. The reference point is marked as r_2 and is chosen such that the region that is formed covers all design points. The point r_1 is obtained from the maxima and minima of the Pareto data. We denote the hypervolume thus obtained from the Pareto front P_n with $\mathcal{V}(P_n)$.

Solutions in the set $\hat{P}_n$ are suggested to be Pareto optimal, and therefore, evaluated to obtain precise area and throughput measurements. Thus, any predicted Pareto front $\hat{P}_n$ is dominated by P_n. Moreover, $\mathcal{V}(P_n) - \mathcal{V}(\hat{P}_n)$ is never negative and can serve as quality measure of the approximation. Further, $\hat{P}_n = P_n$ if and only if $\mathcal{V}(\hat{P}_n) = \mathcal{V}(P_n)$. Fig. 4(b) shows both, the front and the prediction. The difference of the volumes is light blue.

A problem with the hypervolume is that it assesses absolute errors rather than relative ones. This means errors for smaller or slower designs are disproportionately underaccounted for. A more desirable measure would count errors proportionally. It is straightforward to show that this is achieved by calculating the hypervolume in a log-log scale, which converts absolute differences into relative ones on both scales. We call this measure logarithmic hypervolume and denote it with $\mathcal{L}(P_n)$. Fig. 4(c) shows the logarithmic hypervolumes corresponding to Fig. 4(b).

Variable substitution in the integral computing the logarithmic hypervolume shows that it is equivalent to a weighted hypervolume as proposed in [16] with a weighting function of $1/(at)$.

5.2 Pareto Front Percentage Error

Based on the quality indicator $\mathcal{L}$, we define the following metrics to compare an approximation $\hat{P}_n$ with P_n:

$$\mathcal{E}_a(\hat{P}_n) = (1 - e^{\frac{\mathcal{L}(P_n) - \mathcal{L}(\hat{P}_n)}{t'_1 - t'_2}}) \cdot 100, \tag{2}$$

$$\mathcal{E}_t(\hat{P}_n) = (1 - e^{\frac{\mathcal{L}(P_n) - \mathcal{L}(\hat{P}_n)}{a'_1 - a'_2}}) \cdot 100 \tag{3}$$

where $r'_1 = (a'_1, t'_1)$ and $r'_2 = (a'_2, t'_2)$ are the reference points in the log-scaled objective space. $\mathcal{E}_a$ is the average relative distance between P_n and $\hat{P}_n$ along the axis defined by the objective function $\mathcal{A}$; $\mathcal{E}_t$ is the average relative distance between P_n and $\hat{P}_n$ along the axis defined by our objective function $\mathcal{T}$.

The metrics $\mathcal{E}_a$ and $\mathcal{E}_t$ provide an intuitive way to compare approximations $\hat{P}_n$ at different stages of training. Since it is a percentage value and it does not depend on the ranges covered

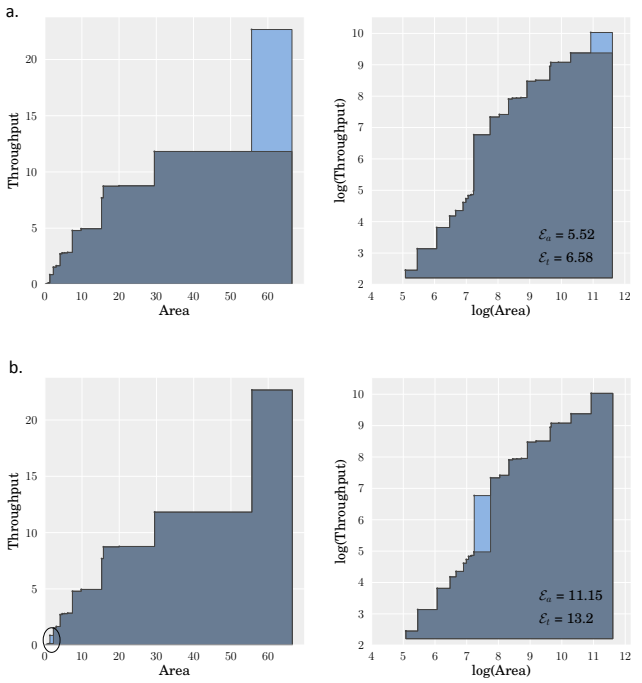

Figure 5. Two examples of Pareto front predictions $\hat{P}_n$ (dark gray region) versus the actual Pareto front P_n (light blue region), in original and log scale, including the corresponding Pareto front percentage errors: (a) $\mathcal{E}_a = 5.52$, $\mathcal{E}_t = 6.58$; (b) $\mathcal{E}_a = 11.15$, $\mathcal{E}_t = 13.2$.

by the volume indicators, it allows for direct comparisons between predictions of different design spaces D_n.

Fig. 5 shows two examples of solutions found by evaluating the configurations predicted by our model to be Pareto (points covered by the gray region), against the Pareto optimal solutions found after exhaustively evaluating the design space (points covered by the blue region). The second plot of (a) and (b) shows the corresponding values of the Pareto front percentage errors ($\mathcal{E}_a$ and $\mathcal{E}_t$). Example (b) shows that what appears as a minor misprediction on the original scale (left plot, circled area) is captured by the metrics $\mathcal{E}_a$ and $\mathcal{E}_t$, which measure the error proportionally.

6. Experimental Evaluation

We evaluate our machine learning algorithm from Section 4 using the IP generators for discrete Fourier transforms (DFTs) and sorting networks (SNWs) (see Section 2). Both offer a large design space characterized by a heterogeneous set of parameters. In the remainder of this section we first explain the experimental setup and then we perform the same set of experiments with both generators:

- We assess the quality of the Pareto front prediction versus the size of training set sampled using the measures $\mathcal{E}_a$ and $\mathcal{E}_t$.
- We determine the accuracy of the generated area and throughput models $\hat{A}$ and $\hat{T}$ on the predicted Pareto set.
- We explore the effect of choosing the threshold in the stopping criterion.

Experimental setup. As explained in Section 2, for every input size $n \in N$ there is a design space D_n of admissible configurations c supported by the generator. For DFTs, we considered $N = \{2^4, \ldots, 2^{14}\}$, and for SNWs $N = \{2^4, \ldots, 2^{14}\}$. Table 1 shows the number of designs $|D_n|$ for every n and the total number $|D|$ of designs.[5]

In both experiments, we evaluated the design space D exhaustively, which took several weeks on a system with 12 cores and 144GB of RAM. All Verilog descriptions were synthesized and place-and-routed for the Xilinx Virtex-6 XC6VLX760 FPGA using Xilinx ISE 13.1. All designs were generated to process 16-bit fixed-point data. Through the exhaustive evaluation we could determine each P_n, $n \in N$, and thus could compute the measures $\mathcal{E}_a$ and $\mathcal{E}_t$ defined in Section 5.2.

We initialized all models by sampling a random 1% of the design space; then samples were "smartly" selected (Alg. 3) until 40% of the design space was explored. Because of the random initialization of the models, the same experiment was repeated 40 times. This way, we can provide averages and their 95% confidence intervals over these 40 experiments.

We evaluate our regression technique based on GPs with our Pareto UCB sampling strategy and refer to it as GP-PUCB. For comparison, we also run the same experiments for GP models with random sampling, referred to as GP-R, and for regression trees (provided by Matlab 7.11) with random sampling, referred to as RT-R.

Pareto front prediction. We evaluate the quality of the Pareto front predictions $\hat{P}_n$ found with increasing size of the training set from 2% up to 40% of $|D|$, in increments of 1%. We measure $\mathcal{E}_a$ and $\mathcal{E}_t$ and average over all $n \in N$. The results are shown in Fig 6: (a) and (b) for SNWs, and plots (c) and (d) for DFTs. The x axes in both plots show the percentage of the design space that has been measured and used for training. Shaded areas indicate 95% confidence intervals. As expected, the errors decrease with increasing training set size.

Fig. 6(a–b) show that, for SNWs, GP-PUCB systematically outperforms RT-R for any percentage of the design space sampled. It also yields considerable gains compared to GP-R once more than about 7% of the training set are sampled. As the training set size increases, GP-PUCB gets considerably closer to the actual Pareto fronts (2% versus 5–7% for the others).

Fig. 6(c–d) show the equivalent results for DFT. A similar behavior can be observed but is less pronounced. Here DT-R is competitive up to sampling 10% of D. Beyond that GP-PUCB is again consistently best.

We conclude that for these IP generators, our method offers a useful prediction, that the GP-based prediction can improve considerably over regression trees (DT-R), and that our smart sampling improves over random sampling.

Area/throughput prediction for Pareto-optimal points. Accurately predicting the Pareto front is not the same as accurately predicting area and throughput separately. The latter implies the former but not vice-versa. For example, the area/throughput predictions may be off but produce a shape similar to the actual front and thus predict the Pareto-points properly.

In the next experiment we show that our method, however, does produce reasonable models $\hat{\mathcal{A}}$ and $\hat{\mathcal{T}}$, which are of interest in their own right. Since our method focuses the exploration on the Pareto region, we show the accuracy of $\hat{\mathcal{A}}$ and $\hat{\mathcal{T}}$ on the predicted Pareto fronts $\hat{P}_n$, $n \in N$. The relative errors in area predictions are shown in Fig. 7(a) and (c); the relative errors in throughput predictions are shown in Fig. 7(b) and (d). The x-axes in all the plots show the percentage of the design space that has been evaluated. Area and throughout are considerably better predicted with GP-PUCB except for a few regions in which GP-R is also competitive. DT-R performs overall poorly.

[5] The numbers $|D_n|$ fluctuate since they depend on the divisibility of n.

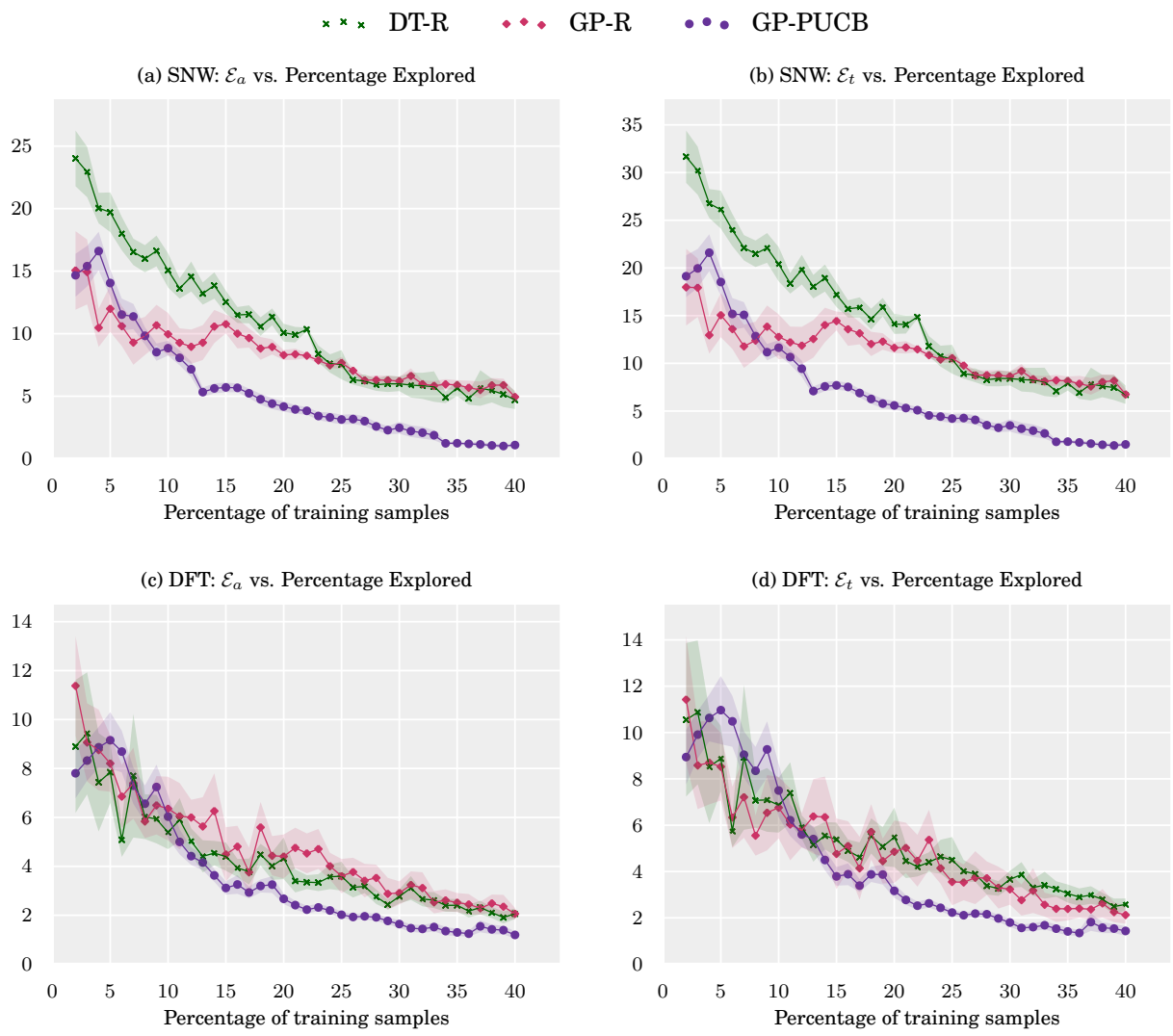

Figure 6. Quality of the Pareto front predictions $\hat{P}_n$, n in N, as measured with $\mathcal{E}_a$ and $\mathcal{E}_t$ versus size of the training set: (a–b) for SNW and (c–d) for DFT. The shaded areas are confidence intervals.

	n	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	all n
$\lvert D_n \rvert$	DFT	20	17	46	30	74	46	58	33	91	30	45	409
	SNW	76	91	148	145	207	100	86	184				1037

Table 1. Design space characterization for DFT and SNW.

Area and throughput predictions with an error of only 10–20% can be achieved with GP-PUCB, while sampling only 10–20% of the design space.

Stopping criterion. So far, we did not invoke our stopping criterion, determined by the threshold parameter h. Next, we test the effect of this parameter with values from 0.01 to 0.3 in steps of 0.01.

Fig. 8 shows, for different values of h, $\mathcal{E}_a$ and $\mathcal{E}_t$ when the stopping condition is met (when $p_2 \leq h$). As expected, the errors decrease with h. Similarly, Fig. 9 shows, for different values of h, the percentage of the design space used for training when the stopping condition is met. As expected, the percentage increases with decreasing h. In this experiment, we limited the percentage explored to 10% to make sure it terminates for very small h, which explains the plateau for $h < 0.05$.

When h is large, the stopping condition is easily met, $\mathcal{E}_a$ and $\mathcal{E}_t$ values are large (see Fig. 8), but a small training set is required (See Fig. 9). On the other hand, when h is small, more training samples are required and better predictions can be made.

Since evaluations are expensive, it is not desirable to evaluate more designs if no improvements in the predictions are observed. For example, from $h = 0.12$ to $h = 0.01$ there are nearly no improvements in the predicted Pareto front. However, the demand for training samples in this range increases up to the limit of 10%. Therefore, a good tradeoff here would be $h = 0.1$. For DFT, the Pareto front would be predicted with errors of 6% and 7% ($\mathcal{E}_a$ and

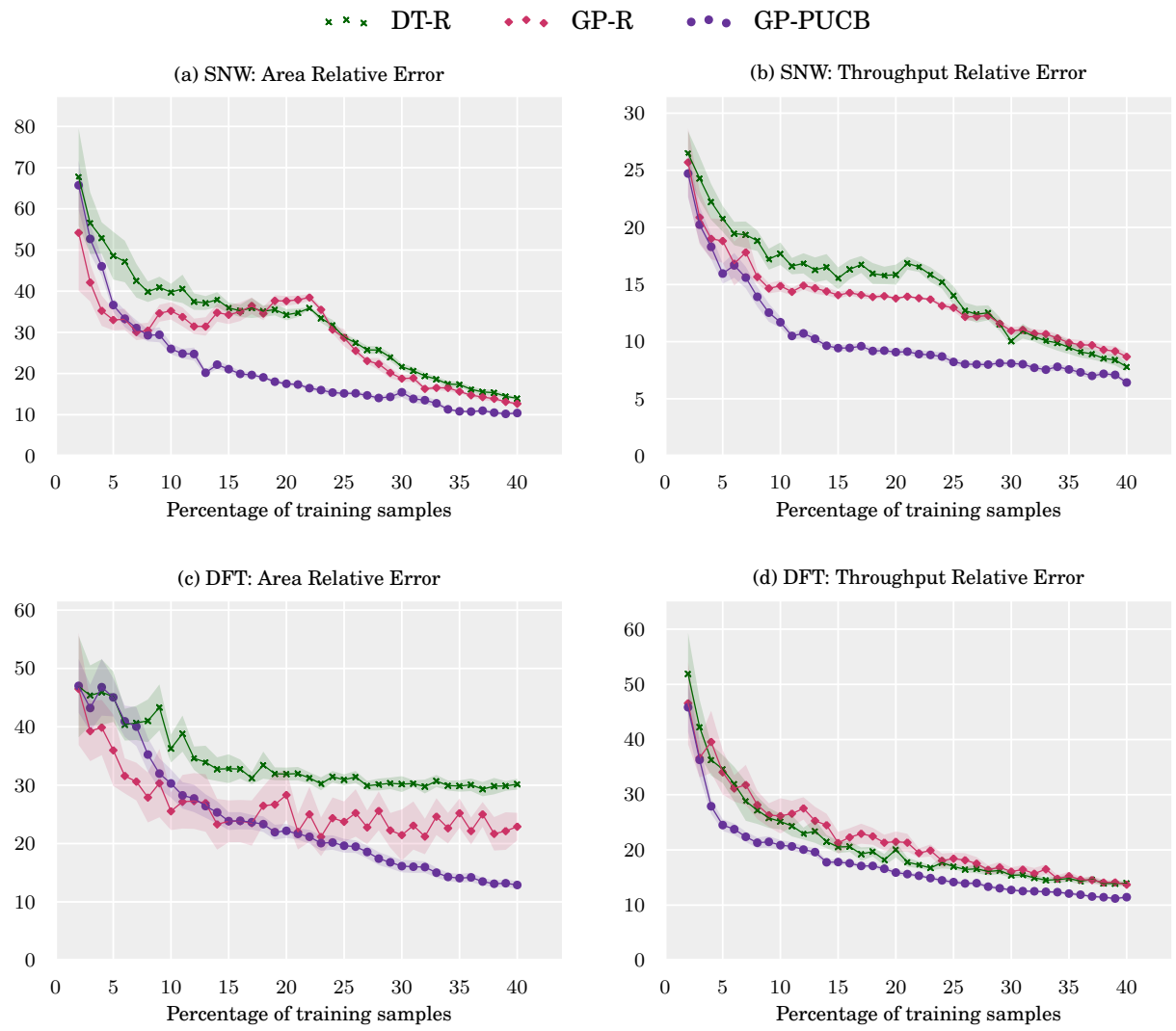

Figure 7. Relative error of throughput and area, calculated for Pareto optimal points, versus size of the training set: (a–b) for SNW and (c–d) for DFT. The shaded areas are confidence intervals.

$\mathcal{E}_t$) with only 6% of the design space evaluated. For SNW, 10% and 13% ($\mathcal{E}_a$ and $\mathcal{E}_t$) error would be obtained by evaluating 3% percent of the design space.

7. Related Work

Multi-objective optimization. Multiple approaches for approximating the Pareto surface of a multi-dimensional objective space have been proposed. Evolutionary algorithms, being one of the most popular of these approaches, have proven to be robust and powerful search mechanisms for tackling the exploration of highly complex design spaces. These algorithms aim at evolving a population to converge to Pareto solutions by emulating natural evolution, supported by concepts such as fitness, elitism, and mutation. The multi-objective nature of the problem raises several challenges to these approaches. Recent works on this topic aim at overcoming these challenges; such as maintaining a diverse population, and defining appropriate fitness functions to suit the multiple objectives [1, 6, 18]. A subset of evolutionary algorithms have used Gaussian processes in order to model the objective functions, and thus eliminating weak candidates from the current population [4, 5]. The main objective of these approaches is to search through the design space until the current population is a good enough approximation of the Pareto set. In contrast, we are interested in extracting a few samples from the design space to predict the Pareto-optimal designs without having to evaluate them during the training stage.

A simplistic approach to a multi objective optimization problem reduces the dimensionality of the objective space by aggregating multiple objectives into a single one, reducing the problem into a simpler one in which a large range of mechanisms can be used, such as statistical inference or single objective evolutionary algorithms. Zhang et al. [15] propose a multiobjective evolutionary algorithm framework that decomposes the optimization problem into single-objective subproblems. A predictive model based on Gaussian processes is built for every subproblem, and sample candidates are selected based on their expected improvement.

Electronic design. Palermo et al. [9] tackle the problem of finding the set of Pareto optimal architectural configurations for multiprocessor systems, while minimizing the number of system-level simulations. First, "design of experiments" techniques are used to initialize a set of training samples. Then response surface model-

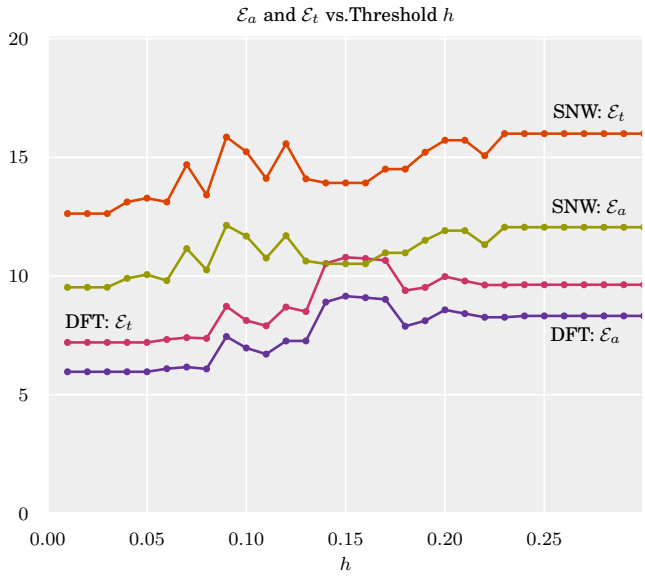

Figure 8. $\mathcal{E}_a$ and $\mathcal{E}_t$ when p_2 equals the threshold h, for different values of h. Better predictions are made as h gets smaller.

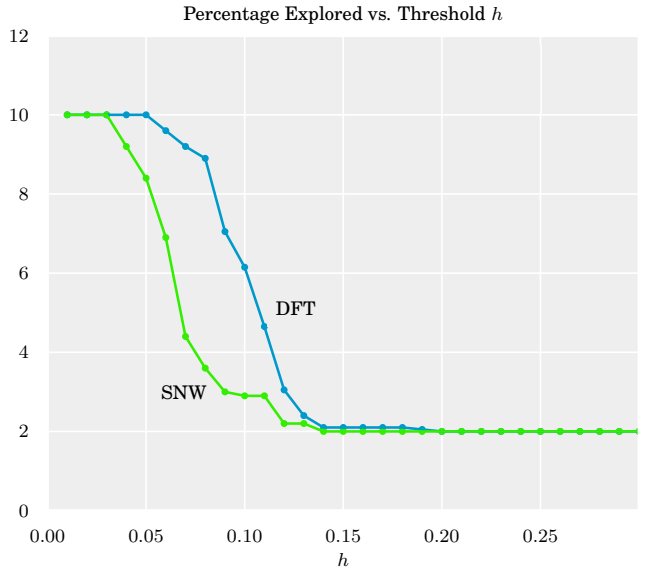

Figure 9. Percentage of the design space explore when p_2 equals the threshold h, for different values of h. Better predictions are made as h gets smaller.

ing techniques are used to predict the objective functions on the remaining configurations. Pareto optimal solutions are taken from the predicted space and then evaluated. This process is repeated until no more improvements are obtained from the simulated configurations or until a maximum number of evaluations is met.

Similarly, our approach to finding the Pareto optimal configurations given a design space, is to model the objective functions and predicting the Pareto configurations based on the predicted space. However, as we are interested in both, estimating and optimizing the objective functions, we explore the design space while targeting potential optimal designs. To achieve this, we build on the Gaussian process upper confidence bound algorithm [13], which only addresses single-objective problems, to balance exploration and exploration in a multi-objective scenario.

In the context of FPGA design, So et al. [12] implemented a high-level synthesis framework where a parallel compiler technology is used to transform application kernels before mapping them into RTL descriptions. The algorithm determines whether a loop should be unrolled and by how much. The tools find an optimal solution by searching over the space of configurations. This search is guided by a set of observations and assumptions about the target applications, and about the impact of unrolling factors on area and performance of the resulting design.

Machine learning techniques, such as non-linear regression and curve fitting, have been already used for area and performance estimation of FPGA designs. However, these are mostly constrained to specific types of implementations. Deng et al. [2] model designs that are composed of a fixed set of supported IP cores. Moreover, it uses a different modeling technique for targeting floating point or fixed point data. Yan et al. [14] target coarse-grained reconfigurable architectures and very long instruction word architectures, while Milder et al. [7] propose area models for a parameterized implementation of DFT, similar to the one considered in this paper. In contrast, our work aims at capturing high-level features of the design, allowing the models to capture the particular patterns of the target application. This ensures that our techniques can be used in a wide range of applications without any modifications.

8. Conclusions

This paper has presented a novel and general machine-learning technique that can be used to find Pareto-optimal implementation tradeoffs in design spaces in which the evaluation of design points is very expensive. The technique is specifically designed for efficiency, i.e., to minimize the number of designs to be synthesized and evaluated using a "smart" sampling strategy. One application is in the context of high-level synthesis tools where parameterization produces a large set of functionally equivalent design points with different cost/performance tradeoffs. We used one such class of tools, fixed-function IP generators for DFTs and sorting networks, for our evaluation. The results show that our method can produce useful predictions of the Pareto front while only sampling a fraction of the design space. Further, our sampling strategy systematically outperforms random sampling for a large enough training set.

Our algorithms have been devised to tackle multi-objective optimization as a general problem. However, more evaluation is needed to assess and understand the benefits of our algorithm in other scenarios and in larger design spaces.

References

[1] C. Coello, G. B. Lamont, and D. Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[2] L. Deng, K. Sobti, and C. Chakrabarti. Accurate Models for Estimating Area and Power of FPGA Implementations. In *Proc. of International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1417–1420, 2008.

[3] M. Ehrgott and X. Gandibleux. A Survey and Annotated Bibliography of Multiobjective Combinatorial Optimization. *OR Spektrum*, (22):425–460, 2000.

[4] M. Emmerich, K. Giannakoglou, and B. Naujoks. Single- and Multiobjective Evolutionary Optimization Assisted by Gaussian Random Field Metamodels. *IEEE Transactions on Evolutionary Computation*, 10(4):421 –439, 2006.

[5] J. Knowles. ParEGO: a Hybrid Algorithm with On-line Landscape Approximation for Expensive Multiobjective Optimization Problems.

IEEE Transactions on Evolutionary Computation, 10(1):50 – 66, 2006.

[6] S. Künzli, L. Thiele, and E. Zitzler. Modular Design Space Exploration Framework for Embedded Systems. *IEE Proceedings Computers & Digital Techniques*, 152(2):183–192, 2005.

[7] P. A. Milder, M. Ahmad, J. C. Hoe, and M. Püschel. Fast and Accurate Resource Estimation of Automatically Generated Custom DFT IP Cores. In *Proc. of International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 211–220, 2006.

[8] P. A. Milder, F. Franchetti, J. C. Hoe, and M. Püschel. Formal Datapath Representation and Manipulation for Implementing DSP Transforms. In *Proc. of Design Automation Conference (DAC)*, pages 385–390, 2008.

[9] G. Palermo, C. Silvano, and V. Zaccaria. ReSPIR: A Response Surface-Based Pareto Iterative Refinement for Application-Specific Design Space Exploration. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(12):1816 –1829, dec. 2009.

[10] C. Rasmussen and H. Nickisch. Gaussian Process Regression and Classification Toolbox Version 3.1 for Matlab 7.x, 2010.

[11] C. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[12] B. So, M. W. Hall, and P. C. Diniz. A Compiler Approach to Fast Hardware Design Space Exploration in FPGA-Based Systems. In *Proc. of Programming Language Design and Implementation (PLDI)*, pages 165–176, 2002.

[13] N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. In *Proc. of International Conference on Machine Learning (ICML)*, 2010.

[14] L. Yan, T. Srikanthan, and N. Gang. Area and Delay Estimation for FPGA Implementation of Coarse-Grained Reconfigurable Architectures. In *Proc. of Languages, Compilers, and Tools for Embedded Systems*, pages 182–188, 2006.

[15] Q. Zhang, W. Liu, E. Tsang, and B. Virginas. Expensive Multiobjective Optimization by MOEA/D with Gaussian Process Model. *IEEE Transactions on Evolutionary Computation*, 14(3):456 –474, 2010.

[16] E. Zitzler, D. Brockhoff, and L. Thiele. The Hypervolume Indicator Revisited: on the Design of Pareto-compliant Indicators via Weighted Integration. In *Proc. of the 4th International Conference on Evolutionary Multi-criterion Optimization (EMO)*, pages 862–876, 2007.

[17] E. Zitzler and S. Künzli. Indicator-based Selection in Multiobjective Search. In *Proc. of the 8th International Conference on Parallel Problem Solving from Nature*, pages 832–842, 2004.

[18] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In *Evolutionary Methods for Design, Optimisation, and Control*, pages 95–100, 2002.

[19] M. Zuluaga, P. Milder, and M. Püschel. Computer Generation of Streaming Sorating Nnetworks. In *Proceedings of the 45th Annual ACM/IEEE Conference on Design Automation (DAC)*, 2012.