On Learning-Based Methods for Design-Space Exploration with High-Level Synthesis

Hung-Yi Liu and Luca P. Carloni
Department of Computer Science, Columbia University, New York, NY, USA
{hungyi, luca}@cs.columbia.edu

ABSTRACT

This paper makes several contributions to address the challenge of supervising HLS tools for design space exploration (DSE). We present a study on the application of learning-based methods for the DSE problem, and propose a learning model for HLS that is superior to the best models described in the literature. In order to speedup the convergence of the DSE process, we leverage *transductive experimental design*, a technique that we introduce for the first time to the CAD community. Finally, we consider a practical variant of the DSE problem, and present a solution based on *randomized selection* with strong theory guarantee.

Categories and Subject Descriptors
B.6.3 [**Design Aids**]: Automatic Synthesis

General Terms
Algorithms, Design, Performance

Keywords
System-Level Design, High-Level Synthesis.

1. INTRODUCTION

It has been a longtime dream that Electronic-System-Level (ESL) design can be automatically synthesized from high-level specifications (e.g. C/C++ or SystemC) to optimized low-level implementations (e.g. RTL or gate-level netlists), a methodology known as High-Level Synthesis (HLS). After many years of endeavor and evolution [9], modern HLS tools can now also take micro-architecture choices as input constraints. By elaborating different sets of constraints, HLS tools allow designers to evaluate multiple implementation alternatives, a process known as Design Space Exploration (DSE). DSE with HLS is already a major leap from DSE with Logic Synthesis, since the latter starts from design specifications given at a lower-level of abstraction using Verilog or VHDL. These hardware-description languages make it more difficult and time-consuming for designers to specify many substantially different micro-architectures.

The industrial adoption of HLS tools is, however, still at an evaluation stage [9]. One of the major bottlenecks is that DSE with HLS still requires substantial efforts for setting the micro-architecture constraints, whose cardinality in general grows exponentially with the size of real design. Another bottleneck is the long runtime of HLS tools: in fact, a simple Discrete Fourier Transform (DFT) design in SystemC may still require an hour of CPU time for one HLS run to complete on a modern computer. Combined, these two challenges are a major roadblock towards the realization of the dream of automatic ESL design.

We make several contributions to address these critical challenges:

- We approach the DSE problem from a machine-learning viewpoint. We propose *Random Forest* [2], a learning model for HLS that is superior to the best known models.
- We present new methods to apply a state-of-the-art DSE framework, which has been proposed for processor design [8, 11, 15] and IP-block macro generators [17], to the context of HLS. Our methods consist of more accurate learning models particularly tailored for HLS. Moreover, for the *first* time in the CAD community, we introduce Transductive Experimental Design (TED) [16], which can judiciously sample *representive* and *hard-to-predict* micro-architecture choices, and use them for training the learning models.
- For complex high-level specifications, we point out a major scalability issue of the existing DSE framework [8, 11, 15, 17], and propose novel and scalable algorithms, which are inspired by recent advancement of machine-learning theory.

2. RELATED WORK

Due to the large solution space, general DSE algorithms rely on local-search techniques, e.g., Genetic Algorithms [7] or Simulating Annealing [13]. When applied with CAD tools, however, these algorithms require at every step actual simulation/synthesis to acquire solution qualities, thereby still suffering from long simulation/synthesis runtime.

Learning-based methods were proposed to guide the DSE process by predicting solution qualities before running actual simulation/synthesis [1, 8, 10, 11, 17]. Compared with local-search techniques, learning-based methods can yield better solution quality as well as require shorter simulation/synthesis runtime. Among these methods, Markov Decision Process was adopted in [1], but this approach may not scale well as it intrinsically traverses an exponentially-growing state space. Alternatively, an efficient framework based on *iterative refinement* was first presented in [11]. Within this framework, two very recent papers [8, 17] independently reported that Gaussian Process [17], a.k.a. Kriging [8], was the most promising learning model, superior to Artificial Neural Network [10, 11] and other simple models. These results were obtained from DSE with *processor simulators* [8] or *IP generators* [17].

In the context of DSE with HLS, learning-based methods were just adopted recently [3, 15]. These works still rely on either local-search techniques [3] or common learning models [3, 15]. In contrast, our work proposes a novel model particularly for HLS, and presents an effective model-training scheme. Moreover, with theory guarantee, our DSE methods are scalable for complex ASIC design.

DAC '13, May 29 - June 07 2013, Austin, TX, USA.

Table 1: Some Common HLS Knobs with Settings

Knob	Setting
Loop Manipulation	Breaking, Unrolling, Pipelining
State Insertion	Adding State Registers
Array Implementation	Registers, Embedded Memory
Function Inlining	Yes or No

3. PROBLEM FORMULATION

We refer to the directives used in a HLS script for determining micro-architecture choices as *knobs*. Table 1 lists some common knobs and their settings available in modern HLS tools. Depending on the knob types, the choices of each knob setting can vary. For example, while the choices of function inlining is binary (either to inline or not to), loop manipulation offer many alternative choices. Let us denote by c the max number of choices of a knob setting.

Given a design specified in high-level languages, e.g. C or SystemC, HLS tools generate an optimized RTL by taking a set of knob settings as input constraints. For a given design specification, the number of places where HLS knobs can be applied can vary and is generally large. For instance, even a small SystemC design may have many functions, many (nested) loops, many arrays, and so on. Let p denote the number of knob-applicable places. Then the total combination of knob settings grows exponentially ($O(c^p)$).

The huge knob-setting space makes DSE with HLS very different from DSE with processor simulators or IP generators. In the latter cases, p can be assumed a small constant in practice. For example, for processor simulators, only a limited set of parameters needs to be determined, e.g., the number of cores, the size of L1/L2 cache, etc., and similarly for IP generators, for which the parameters are the I/O size, algorithms, etc. Instead, p is typically a large constant for DSE with HLS, which therefore becomes an even more challenging problem. Since to exhaustively explore the set of of all possible RTL designs that can be obtained with HLS is unfeasible, the goal of DSE with HLS is to derive an approximation of the set of Pareto-optimal designs.

We consider the *DSE with HLS* problem as follows.

PROBLEM 1. *Given a high-level design specification and HLS tool with a budget of b runs, find the best approximate Pareto-optimal set of RTL designs without exceeding b.*

Note that we consider a multi-objective DSE problem. Although throughout this paper we focus on 2-objective cases for simplicity, all our discussions and findings are generally applicable to higher dimensional cases.

In order to measure the quality of an approximate set of Pareto-optimal designs, we utilize the metric of *average distance from reference set* (ADRS) [11]. Consider a two-dimensional (area $\mathcal{A}$ vs. effective-latency $\mathcal{T}$) design space[1]. For both objectives, the smaller the objective, the better the RTL implementation. Given a reference Pareto set $\Pi = \{\pi_1, \pi_2, \dots | \pi_i = (a,t), a \in \mathcal{A}, t \in \mathcal{T}\}$ and an approximate Pareto set $\Lambda = \{\lambda_1, \lambda_2, \dots | \lambda_j = (a,t), a \in \mathcal{A}, t \in \mathcal{T}\}$,

$$\text{ADRS}(\Pi, \Lambda) = \frac{1}{|\Pi|} \sum_{\pi \in \Pi} \min_{\lambda \in \Lambda} \delta(\pi, \lambda),$$

where

$$\delta(\pi = (a_\pi, t_\pi), \lambda = (a_\lambda, t_\lambda)) = \max\{0, \frac{a_\lambda - a_\pi}{a_\pi}, \frac{t_\lambda - t_\pi}{t_\pi}\}.$$

Note that the lower $\text{ADRS}(\Pi, \Lambda)$, the closer the approximate set Λ to the reference set Π.

[1] We define the *effective latency* as the product of the clock period and the clock cycle count.

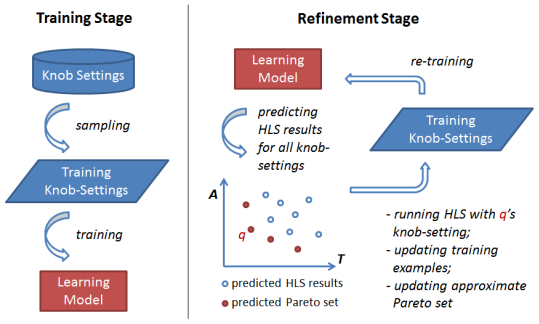

Fig. 1: The iterative-refinement DSE framework [8, 11, 15, 17].

Algorithm 1 Iterative-Refinement Framework

Input: HLS tool H, HLS budget b, input design D
Output: Approximate Pareto set Λ

1: Let $\mathcal{K}$ be the knob-setting space of D
2: Let $\tilde{H}$ be a learning model
3: Let $\tilde{\mathcal{K}} \subset \mathcal{K}$ be a training set
4: Let $S = \phi$ be the set of all HLS results
5: **/** Training Stage **/**
6: synthesize all $\tilde{\mathbf{k}} \in \tilde{\mathcal{K}}$; add the results to $\mathcal{S}$
7: remove $\tilde{\mathcal{K}}$ from $\mathcal{K}$
8: train $\tilde{H}$ by $(\tilde{\mathcal{K}}, \mathcal{S})$
9: **/** Refinement Stage **/**
10: $\Lambda \leftarrow$ current Pareto approximation of $\mathcal{S}$
11: **for** $i = |S| + 1 \rightarrow b$ **do**
12: $\tilde{\mathcal{Q}} \leftarrow$ predicted HLS results $\forall \mathbf{k} \in \mathcal{K}$ by $\tilde{H}$
13: pick one $q \in \tilde{\mathcal{Q}}$ for HLS; add q's result to $\mathcal{S}$
14: move q's knob setting from $\mathcal{K}$ to $\tilde{\mathcal{K}}$
15: retrain $\tilde{H}$ by $(\tilde{\mathcal{K}}, \mathcal{S})$
16: $\Lambda \leftarrow$ current Pareto approximation of $\mathcal{S}$
17: **end for**

4. PRELIMINARY

We start with a case study of DFT designed in SystemC. The DFT design implements an iterative-FFT algorithm [5] with a synthesizable fix-point library using 45nm technology. The total number of knob-setting combinations for the DFT design is 360,448. In order to establish a ground truth for the analysis of Pareto optimality, we synthesized a restricted set of 242 knob settings, which cover 11 loop manipulations, 2 function-inlining choices, and 11 feasible clock periods. The 242 HLS runs took an aggregated 84-hour CPU time with a commercial HLS tool. Throughout this paper, we refer to the 242 knob settings as the knob-setting space of the DFT.

Next we review a state-of-the-art DSE framework [8, 11, 15, 17], which is also illustrated in Fig. 1. The main idea of the framework is twofold: *(i)* to approximate the HLS tool H by a learning model $\tilde{H}$, and *(ii)* to use the fast $\tilde{H}$ for predicting the quality-of-result space, instead of invoking the time-consuming H. Algorithm 1 describes the major steps of the framework. Initially, the framework trains $\tilde{H}$ by spending some HLS runs (the *training* stage in Lines 6–8), and then finds an approximate Pareto set by iteratively refining $\tilde{H}$ and the current Pareto approximation (the *refinement* stage in Lines 9–16).

5. ALGORITHMS

Clearly, the effectiveness of the iterative-refinement framework relies on the accuracy of $\tilde{H}$. Intuitively, guided by a highly-accurate $\tilde{H}$, the refinement stage could search in the right space. However,

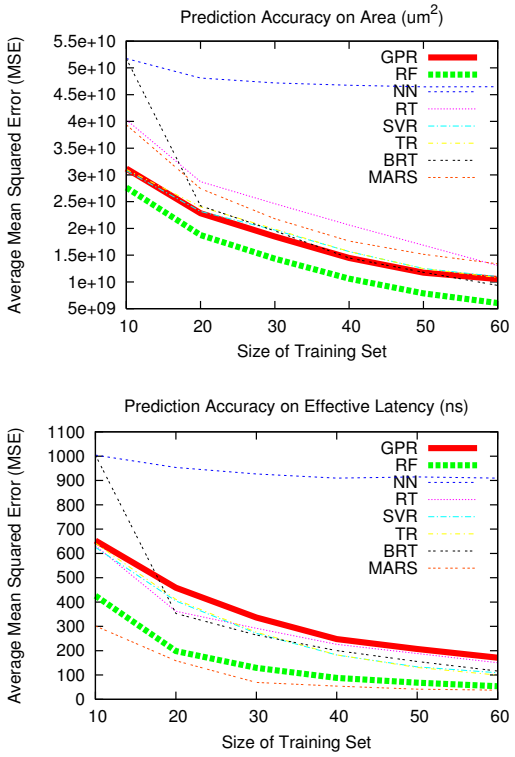

Fig. 2: Learning-model accuracy for predicting DFT area (top) and effective latency (bottom). The training sets are randomly sampled.

spending too many HLS runs in the training stage while aiming for highly-accurate H, may not necessarily guarantee the final Pareto optimality, since the total number of HLS runs is limited to a given budget b. We approach this dilemma by suggesting a more accurate learning model for HLS in Section 5.1 and a more effective training scheme that is beneficial for general models in Section 5.2. Moreover, in Sections 5.3 and 5.4, we identify a scalability issue of the framework, and propose novel algorithms. All our algorithmic findings are inspired by recent machine-learning theory.

5.1 Learning Models for HLS

We examined eight advanced learning models for predicting the DFT's area and effective latency. These models include Gaussian Process Regression (GPR), Random Forest (RF), Neural Network (NN), Regression Tree (RT), Support Vector Regression (SVR), Transductive Regression (TR), Boosted Regression Tree (BRT), and Multivariate Adaptive Regression Splines (MARS)[2]. Note that we examined five of these models (namely RF, SVR, TR, BRT, and MARS) for the first time in the DSE literature.

Fig. 2 shows the prediction accuracy of these models that were trained on randomly-sampled training sets[3]. The results suggest that RF is consistently more accurate than GPR, which was previously reported as the best model for processor simulators and IP generators. The adoption of RF for HLS instead of GPR brings the following benefits.

[2] All these models are publicly available in separate R packages [12], except TR [6], which we implemented in R by ourselves. Notice that we finely tuned all these models via *model selection* [14] to achieve the best accuracy.

[3] Throughout this paper, all the results involving randomized procedures are the average results obtained from 100 trials.

Algorithm 2 Sequential TED

Input: Set $\mathcal{K}$ of n knob settings, training-set size m
Output: Training set $\tilde{\mathcal{K}}$

1: $\mathbf{F} \leftarrow \mathbf{F}_{\mathbf{k},\mathbf{k}}$
2: $\tilde{\mathcal{K}} \leftarrow \phi$
3: **for** $i = 1 \rightarrow m$ **do**
4: select $\mathbf{k}_i \in \mathcal{K}$ with the largest $||\mathbf{F}_{\mathbf{k}_i}||^2/(f(\mathbf{k}_i, \mathbf{k}_i) + \mu)$, where $\mathbf{F}_{\mathbf{k}_i}$ and $f(\mathbf{k}_i, \mathbf{k}_i)$ are $\mathbf{k}_i$'s corresponding column and diagonal entry in current $\mathbf{F}$
5: add $\mathbf{k}_i$ to $\tilde{\mathcal{K}}$,
6: $denom \leftarrow f(\mathbf{k}_i, \mathbf{k}_i) + \mu$
7: $(\mathbf{F})_{jk} \leftarrow (\mathbf{F})_{jk} - \frac{(\mathbf{F})_{ji}(\mathbf{F})_{ki}}{denom}, \forall 1 \leq j, k \leq n$
8: **end for**

- HLS knobs which provide *binary* choices are common, e.g., function inlining for a function call, state insertion for a certain edge in the control/data flow graph, array implementation in either registers or memory, and others. In this regard, GPR assumes that every knob variable follows a Gaussian distribution, which is obviously not proper for the binary-valued knobs. Instead, the tree-based RF model can easily handle binary decisions by introducing a node with two branches separating the two decisions.

- RF is an ensemble model consisting of multiple regression trees [2]. Given a training set, the set is internally and randomly partitioned for training individual trees. Then, the final prediction is made by collective vote from the individual trees. The two steps combined are capable of minimizing both the generalization error and prediction variance. A recent study in the Machine Learning literature also shows the superior accuracy of RF, especially for *high dimensional* data [4].

- We observe that the CPU time required for training and prediction with RF is around 50% less than that with GPR. Moreover, the internal partitioning-then-training scheme by nature makes RF suitable for running on multi-core machines. From an implementation viewpoint, this is another advantage of adopting RF.

Given $\tilde{H}$ being either GPR or RF, we conclude this section by discussing how to select a best knob-setting for next HLS in the refinement stage, i.e., Line 13 in Algorithm 1. **For GPR**, the prediction of a design objective consists of a mean and a variance. The mean represents the predicted value, and the variance suggests the uncertainty of GPR about the prediction. Therefore, as suggested in [17], a *predicted* Pareto set $\tilde{\Lambda}$ is first extracted from the predicted objective space $\tilde{\mathcal{Q}}$ (i.e., Line 12 in Algorithm 1), and then the element $\in \tilde{\Lambda}$ with the max variance (uncertainty) across all objectives is selected for the next HLS. On the other hand, **for RF**, since the prediction uncertainty is minimized by RF's collective-vote scheme, we just randomly pick one element $\in \tilde{\Lambda}$ for next HLS.

5.2 Transductive Experimental Design (TED)

In the prior work [8, 17], the training set $\tilde{\mathcal{K}}$ is randomly sampled from $\mathcal{K}$ (Line 3 in Algorithm 1). Alternatively, we introduce *transductive experimental design* (TED) [16], that aims for selecting *representative* as well as *hard-to-predict* $\tilde{\mathcal{K}}$, in order to effectively train the learning model for predicting $\mathcal{K}$. *Note that TED assumes no priori knowledge about the learning model and should therefore be beneficial for any model.*

Assume that overall we have n knob settings ($|\mathcal{K}| = n$), from which we want to select a training set $\tilde{\mathcal{K}}$ such that $|\tilde{\mathcal{K}}| = m$. In

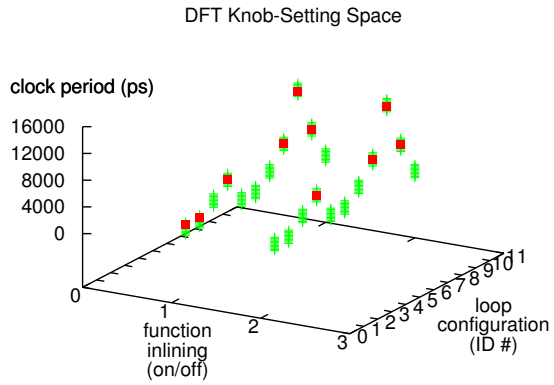

Fig. 3: Training-set sampling by Transductive Experimental Design (TED). See Section 5.2 for details.

order to minimize the prediction error $H(\mathbf{k}) - \tilde{H}(\mathbf{k})$ for all $\mathbf{k} \in \mathcal{K}$, TED is shown to be equivalent to the following problem.

$$\begin{array}{rl} \max_{\tilde{\mathcal{K}}} & T\left[\mathcal{K}\tilde{\mathcal{K}}^\top(\tilde{\mathcal{K}}\tilde{\mathcal{K}}^\top + \mu\mathbf{I})^{-1}\tilde{\mathcal{K}}\mathcal{K}^\top\right] \\ \text{s.t.} & \tilde{\mathcal{K}} \subset \mathcal{K}, |\tilde{\mathcal{K}}| = m, \end{array} \quad (1)$$

where $T[\,]$ is a matrix trace and $\mu > 0$ given. The solution to the problem can be interpreted as follows. It tends to find *representative* data samples $\tilde{\mathcal{K}}$ that span a linear space to retain most of the information of $\mathcal{K}$ [16].

Equation 1 corresponds to TED sampling for *linear* regression. For *non-linear* regression, TED can be extended by a kernel function

$$f(\mathbf{u}, \mathbf{v}) = \theta(\mathbf{u}) \cdot \theta(\mathbf{v}),$$

where $\mathbf{u}, \mathbf{v} \in \mathcal{R}^d, \theta : \mathcal{R}^d \rightarrow \mathcal{F}$ is a function mapping from the knob-setting space to a feature space. In this paper, we use Gaussian kernel as in [16]:

$$f(\mathbf{u}, \mathbf{v}) = e^{\frac{||\mathbf{u}-\mathbf{v}||^2}{2\sigma^2}},$$

where σ is a given non-zero constant. Then, kernelized TED can be expressed as

$$\begin{array}{rl} \max_{\tilde{\mathcal{K}}} & T\left[\mathbf{F}_{\mathbf{k}\tilde{\mathbf{k}}}(\mathbf{F}_{\tilde{\mathbf{k}}\tilde{\mathbf{k}}} + \mu\mathbf{I})^{-1}\mathbf{F}_{\tilde{\mathbf{k}}\mathbf{k}}\right] \\ \text{s.t.} & \tilde{\mathcal{K}} \subset \mathcal{K}, |\tilde{\mathcal{K}}| = m, \end{array} \quad (2)$$

where matrix entries $(\mathbf{F}_{\mathbf{k}\tilde{\mathbf{k}}})_{ij} = f(\mathbf{k}_i, \tilde{\mathbf{k}}_j), (\mathbf{F}_{\tilde{\mathbf{k}}\tilde{\mathbf{k}}})_{ij} = f(\tilde{\mathbf{k}}_i, \tilde{\mathbf{k}}_j)$, $(\mathbf{F}_{\tilde{\mathbf{k}}\mathbf{k}})_{ij} = f(\tilde{\mathbf{k}}_i, \mathbf{k}_j)$, vectors $\mathbf{k}_i, \mathbf{k}_j \in \mathcal{K}$, vectors $\tilde{\mathbf{k}}_i, \tilde{\mathbf{k}}_j \in \tilde{\mathcal{K}}$.

Unfortunately, both TED problems (Equations 1 and 2) are proven to be NP-hard [16]. Therefore, we apply an efficient greedy algorithm, *sequential TED* [16], to solve Equation 2. The algorithm (given as Algorithm 2) can be interpreted as follows. Once a best $\mathbf{k}_i \in \mathcal{K}$ is selected (Line 4), the kernel matrix $\mathbf{F}$ is updated (Line 7) to represent the residual knob settings, so that the next selection would be picked among those under-represented by previously selected settings. In other words, the algorithm tends to select knob settings that cannot be well represented by selected ones, i.e., to favor potentially *hard-to-predict* knob settings if not being selected as training examples.

Fig. 3 illustrates the TED sampling results of the DFT knob-setting space. We follow the suggestion in [16] to set $\mu = 0.1$ for Algorithm 2 and $\sigma = 0.1$ for Gaussian kernel. As a result, a training set of 10 knob settings (red squares) is selected from the 242 candidates (green pluses). We can see that TED indeed selects *representative* samples by distributing its selections in the whole space, without having dense clusters. Besides, TED distributes 6 out of the 10 selections to the subspace where the loop-configuration ID number is greater than 6, a subspace where candidates are sparser than its counterpart. The loop configurations are generated by varying one loop-manipulation knob for consecutive configurations. Hence, the knob settings (green pluses) that are close to each other are very likely to produce similar HLS results. Based on this observation, TED considers the samples in the sparse subspace *hard-to-predict*, thereby selecting more samples there.

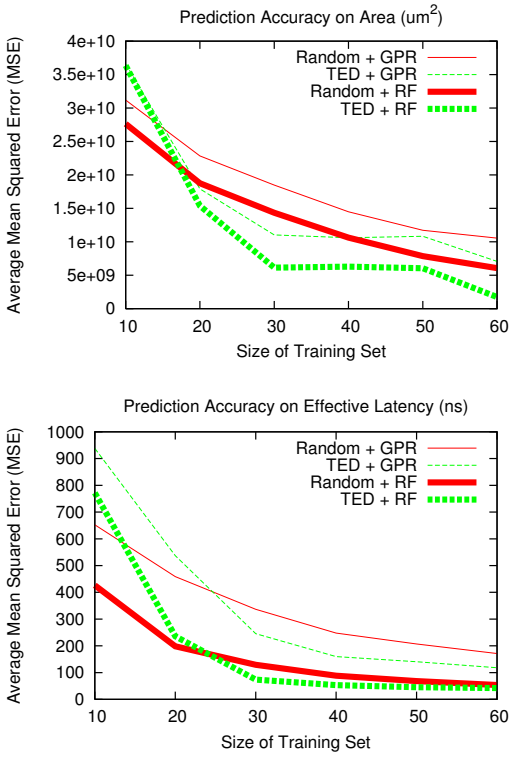

Fig. 4: Learning-model accuracy for predicting DFT area (top) and effective latency (bottom). "Random" and "TED" indicates training-set sampling algorithms. "GPR" and "RF" are learning models.

Fig. 4 shows the prediction accuracy of the combination of random/TED sampling with GPR/RF models. Starting from the sample size of 20 knob-settings for predicting area and 30 knob-settings for predicting effective latency, TED indeed reduces the prediction error for GPR and RF, respectively (see the dashed vs. solid lines). On the other hand, for very small training-set size, random sampling is helpful to escape from local optimal where the sequential-TED algorithm could be trapped. In general, TED is very effective for sampling good training sets for any learning models. Finally, we remark that with the aid of TED sampling, the prediction error of RF is still consistently lower than that of GPR (see the dashed lines in Fig. 4).

5.3 Randomized Selection

The size of knob-setting space $\mathcal{K}$ grows exponentially as explained in Section 3. Therefore, even if a very fast learning model $\tilde{H}$ is used, the exhaustive search in $\mathcal{K}$ (Line 12 in Algorithm 1) makes the iterative-refinement framework not scalable. From now on, we refer to Problem 1 with very large $\mathcal{K}$ as the **Extreme DSE-with-HLS Problem**, and refer to Problem 1 with tractable $\mathcal{K}$ as the **Basic DSE-with-HLS Problem**.

To conquer the extreme DSE-with-HLS problem, we introduce the following theorem on *randomized selection* [14].

Table 2: Comparison of DSE Methods

Method	Reference	Learning Model	Training-Set Sampling	Next-HLS Selection
`state-of-the-art`	[17]	GPR	random sampling	exhaustive search
`basic`	Section 5.1	RF	random sampling	exhaustive search
`basic-ST`	Section 5.2	RF	sequential TED	exhaustive search
`extreme`	Section 5.3	RF	random sampling	randomized selection
`extreme-RT`	Section 5.4	RF	randomized TED	randomized selection

Algorithm 3 Randomized TED

Input: Set $\mathcal{K}$ of n knob settings, training-set size m
Output: Training set $\tilde{\mathcal{K}}$

1: $\tilde{\mathcal{K}} \leftarrow \phi$
2: **for** $i = 1 \rightarrow m$ **do**
3: $\quad \tilde{M} = \{\mathbf{m}_1, \ldots, \mathbf{m}_{N_{rted}}\} \leftarrow$ a random subset of $\mathcal{K}$
4: $\quad \tilde{M} = \tilde{M} \cup \tilde{\mathcal{K}}$; $\mathbf{F} \leftarrow \mathbf{F}_{\mathbf{m},\mathbf{m}}$
5: $\quad \forall \mathbf{m}_i \in \tilde{\mathcal{K}}$, update $\mathbf{F}$ as Lines 6–7 in Algorithm 2
6: $\quad$ select $\mathbf{m}_i \in \tilde{M}$ as Line 4 in Algorithm 2; add $\mathbf{m}_i$ to $\tilde{\mathcal{K}}$
7: **end for**

THEOREM 1 (**Ranks on Random Subsets**). *Let $M = \{x_1, \ldots, x_\alpha\} \subset \mathcal{R}$, and let $\tilde{M} \subset M$ be a random subset of size β. Then the probability that $\max \tilde{M}$ is greater than γ elements of M is at least $1 - (\frac{\gamma}{\alpha})^\beta$.*

According to the theorem, if we draw a random subset $\tilde{M}$ of size 59 ($\beta = 59$), then $\max \tilde{M}$ would be greater than 95% ($\frac{\gamma}{\alpha} = 95\%$) elements of M with at least $1 - 5\% = 95\%$ confidence, since $(\frac{\gamma}{\alpha})^\beta = 0.95^{59} < 5\%$. Note that the sample size 59 is a *constant* for any large-sized M to achieve a 95% confidence in the 95% percentile range.

Based on Theorem 1, we propose a simple modification for selecting the next HLS in Algorithm 1 (Lines 12–13): draw a random subset $\tilde{M} \subset \mathcal{K}$ of size N_{next}, and then pick the $q \in \tilde{M}$ with the smallest

$$\tilde{H}_{\mathcal{A}}(q) + \tilde{H}_{\mathcal{T}}(q), \tag{3}$$

where $\tilde{H}_{\mathcal{A}}(q)$ and $\tilde{H}_{\mathcal{T}}(q)$ are the predicted area and effective latency of q, respectively. Clearly, Theorem 1 also applies to selecting a minimum element. Consequently, we pick the best $q^* \in \tilde{M}$ for HLS, based on the cost function defined as Equation 3, which predicts the quality-of-result of any $q \in \tilde{M}$. Note that to set $N_{next} = 59$ should suffice to achieve a very good approximation even for a very large $\mathcal{K}$, as inferred from Theorem 1.

5.4 Randomized TED

Consider again the prohibitively large size of $\mathcal{K}$ of the extreme DSE-with-HLS problem. This makes even the training-set sampling by TED not scalable, since an $O(|\mathcal{K}|^2)$ matrix computation is required in the sequential-TED algorithm. To address this scalability issue, inspired again by Theorem 1, we propose the *randomized TED* algorithm (as Algorithm 3). The main idea is that in each iteration we draw a random subset $\tilde{M} \subset \mathcal{K}$ of size N_{rted}, and add previously selected samples to $\tilde{M}$. $\tilde{M}$ is now treated as the $\mathcal{K}$ in the sequential-TED algorithm. We update the kernel matrix as if the previously selected samples are selected again in this iteration. Then we follow the same criterion as in the sequential-TED algorithm to select the best residual $\mathbf{m} \in \tilde{M}$, and we iterate until m samples are selected.

Overall, our algorithm utilizes the randomized-selection scheme to reduce the computational cost, while still preserving the principle of TED. Note that since we want to approximate the best m elements, as opposed to the best one only, we should expect the constant N_{rted} to be larger than 59 in order to achieve an equivalent approximation as discussed in Section 5.3.

Table 3: Number of HLS Runs to Find the Exact Pareto Set

Training-Set Size	10	20	30	40	50	60
`state-of-the-art`	120	NA	NA	NA	NA	NA
`basic`	115	113	NA	NA	NA	NA
`basic-ST`	112	91	100	100	109	119

6. EXPERIMENTAL RESULTS

We continue our DFT case study to compare the four DSE methods that we presented in the previous section among them and with respect to the existing method proposed in [17], which we denote as `state-of-the-art`. Table 2 summarizes the five methods. We call `basic` our method that utilizes RF as the learning model, random sampling for selecting training knob-settings, and exhaustive search for selecting the next HLS knob-setting. We call `basic-ST` the variant of `basic` that utilizes sequential TED for selecting training knob-settings. Then, we refer to our method that utilizes RF as the learning model, random sampling for selecting training knob-settings, and randomized selection for selecting the next HLS knob-setting, as `extreme`. And finally we call `extreme-RT` the variant of `extreme` that utilizes randomized TED for selecting training knob-settings.

For each DSE method, we prepared training sets of size $m \in \{10, 20, \ldots, 60\}$. For each value of m, we set a HLS budget $\in \{m+10, m+20, \ldots, 120\}$ and we tracked the average ADRS that each method can achieve, with the exact Pareto set as a reference set. The HLS budget was capped at 120, i.e., less than 50% of DFT's knob-setting space. The results of all the experiments are collected in Table 6, which is placed in the Appendix for space reasons. Note that across all training-set sizes, each method should have $11 + 10 + 9 + 8 + 7 + 6 = 51$ ADRS records.

Results for the basic DSE-with-HLS problem. For the basic problem, we compare three methods: `basic`, `basic-ST`, and `state-of-the-art`.

Table 3 summarizes the total number of HLS runs that each method requires to find the exact Pareto set (i.e., ADRS $= 0$). We can see that `basic-ST`, which utilizes TED to sample training sets, is the only method that can find the exact Pareto set within 120 HLS runs for any training-set size. Besides, for any training-set size, both `basic` and `basic-ST`, which adopt the RF learning model, outperform `state-of-the-art`, which adopts GPR instead. These results confirm that a more accurate learning model with a more effective training-set sampling scheme (in our case, RF with TED) indeed facilitates the convergence of the iterative-refinement framework. Also note that the best training-set size happens at 20, followed by 30 and 40. This result suggests that `basic-ST` only requires small training sets to achieve its best performance, because TED can judiciously select those *representative* and *hard-to-predict* knob-settings as training examples.

As explained in Section 3, in real applications of DSE with HLS, it is infeasible to search for the exact Pareto set. Therefore, we now examine the performance of those three methods given different HLS budgets. For HLS budget $b \in \{20, 30, \ldots, 120\}$, Table 4 summarizes the best DSE method, its average ADRS, and the training-set size it requires. We can observe that for smaller budgets ($20 \leq b \leq 50$), `basic` (abbreviated as `bs` in Table 4) can yield the

Table 4: `state-of-the-art` [17] vs. `basic` (`bs`) vs. `basic-ST` (`bs-ST`) given HLS budget b

Budget b	20	30	40	50	60	70	80	90	100	110	120
Best Method	bs	bs	bs	bs	bs-ST	bs-ST	bs-ST	bs-ST	bs-ST	bs-ST	bs-ST
Average ADRS (%)	21.54	9.01	3.58	1.74	0.98	0.21	0.11	0.01	0.00	0.00	0.00
Training-Set Size	10	20	20	20	30	30	30	20–30	20–40	20–50	>=10

Table 5: `extreme` (`ex`) vs. `extreme-RT` (`ex-RT`) given HLS budget b

Budget b	20	30	40	50	60	70	80	90	100	110	120
Best Method	ex-RT	ex-RT	ex-RT	ex-RT	ex-RT	tie	ex	ex	ex-RT	ex-RT	tie
Average ADRS (%)	19.12	9.23	5.82	3.52	2.00	1.09	0.58	0.17	0.10	0.04	0.01
Training-Set Size	10	20	20	20	20	30/50	40	40	40	40	40

minimum average-ADRS, requiring a training-set of size around 20. For greater budgets ($60 \leq b \leq 120$), `basic-ST` (abbreviated as `bs-ST`) stands out to achieve the minimum average-ADRS, requiring a training-set of size around 30. In general, `basic-ST` starts with a higher ADRS because TED sampling does not favor the knob-settings that can result in Pareto-optimal RTLs, whereas they could be sampled at random instead. However, `basic-ST` can approach the exact Pareto front faster, due to the more accurate learning model resulting from the aid of TED. These two reasons combined explain why `basic` performs better for smaller HLS budgets, whereas `basic-ST` is better for greater budgets. Overall, both our methods, `basic` and `basic-ST`, outperform `state-of-the-art`.

Results for the extreme DSE-with-HLS problem. For the extreme problem, we focus on two methods: `extreme` and `extreme-RT`.

First we compare `extreme` with `basic` to see how effective the approximation using randomized-selection can be. For `extreme`, we set $N_{next} = 59$ for drawing a random subset of size 59 to select the next knob-setting for HLS. As expected, most of the ADRS achieved by `extreme` are higher (worse) than those achieved by `basic`, but we find that the difference is small: for training-set size $10 \leq m \leq 30$, the max difference is less than 3.0%, and for $40 \leq m \leq 60$, the max difference is even less than 0.8%. These small differences show that the approximation using randomized-selection can be very effective. Moreover, since the ADRS differences are marginal, we observed that 41 out of 51 (80%) ADRSs achieved by `extreme` are also lower (better) than those achieved by `state-of-the-art`. For $N_{next} \in \{59, 79, 99\}$, we see no significant difference on ADRS: in fact, 46 out of 51 (90%) ADRS differences are less than 0.9%. Therefore, we set $N_{next} = 59$ by default for `extreme`.

Next, we show the results of `extreme-RT`. For `extreme-RT`, we fix its $N_{next} = 59$ and set $N_{rted} \in \{59, 79, 99\}$ for drawing a random subset of size N_{rted} to select a training knob-setting by the randomized-TED algorithm. Compared with `extreme`, if `extreme-RT`'s $N_{rted} = 59$, we observe that only 17 out of 51 (33%) ADRSs achieved by `extreme-RT` are lower (better). If we increase N_{rted} to 79, then 30 out of 51 (59%) ADRSs are better. The improvement saturates at $N_{rted} = 99$, where 31 out of 51 (61%) ADRSs are better. As a result, the randomized-TED algorithm works best with $N_{rted} = 99$. Therefore, we set $N_{rted} = 99$ by default for `extreme-RT`.

Finally, for the extreme problem with different HLS budgets, Table 5 summarizes the best method, its average ADRS, and the training-set size it requires. We see that `extreme-RT` (abbreviated as `ex-RT`) almost outperforms `extreme` (abbreviated as `ex`) for every budget b. This result is different from the `basic` vs. `basic-ST` result (see Table 4), where `basic` (`basic-ST`) works better for smaller (greater) budgets. This is because the random-subset drawing used in the randomized-TED algorithm can alleviate the high initial ADRS otherwise caused by the sequential-TED algorithm. Overall, we find `extreme-RT` very effective for addressing the extreme problem, because our randomized-TED algorithm successfully avoids the exhaustive search in the full knob-setting space while still selecting *representative* and *hard-to-predict* knob-settings as training examples.

7. CONCLUSIONS

We have presented novel learning-based methods for DSE with HLS. Our methods based on the Random-Forest learning model, transductive experimental design, and randomized selection, can effectively find an approximate Pareto set of RTL designs.

Acknowledgments. This work is partially supported by an ONR Young Investigator Award, the National Science Foundation (Award #1219001), and the DARPA Perfect program.

8. REFERENCES

[1] G. Beltrame, L. Fossati, and D. Sciuto. Decision-theoretic design space exploration of multiprocessor platforms. *IEEE TCAD*, 29(7):1083–1095, July 2010.

[2] L. Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, Oct. 2001.

[3] B. Carrion Schafer and K. Wakabayashi. Machine learning predictive modelling high-level synthesis design space exploration. *Computers Digital Techniques, IET*, 6(3):153–159, May 2012.

[4] R. Caruana, N. Karampatziakis, and A. Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *Proc. of the 25th Intl. Conf. on Machine learning*, pages 96–103, 2008.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[6] C. Cortes and M. Mohri. On transductive regression. In *Advances in Neural Information Processing Systems (NIPS)*, pages 305–312, 2006.

[7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Trans. on Evolutionary Computation*, 6(2):182–197, Apr. 2002.

[8] G. Mariani, G. Palermo, V. Zaccaria, and C. Silvano. Oscar: An optimization methodology exploiting spatial correlation in multicore design spaces. *IEEE TCAD*, 31(5):740–753, May 2012.

[9] G. Martin and G. Smith. High-level synthesis: Past, present, and future. *IEEE Design Test of Computers*, 26(4):18–25, Aug. 2009.

[10] B. Ozisikyilmaz, G. Memik, and A. Choudhary. Efficient system design space exploration using machine learning techniques. In *Proc. of DAC*, pages 966–969, 2008.

[11] G. Palermo, C. Silvano, and V. Zaccaria. Respir: A response surface-based pareto iterative refinement for application-specific design space exploration. *IEEE TCAD*, 28(12):1816 –1829, Dec. 2009.

[12] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012.

[13] B. C. Schafer, T. Takenaka, and K. Wakabayashi. Adaptive simulated annealer for high level synthesis design space exploration. In *Proc. of VLSI-DAT*, pages 106–109, Apr. 2009.

[14] B. Scholkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001.

[15] S. Xydis, G. Palermo, V. Zaccaria, and C. Silvano. A meta-model assisted coprocessor synthesis framework for compiler/architecture parameters customization. In *Proc. of DATE*, 2013.

[16] K. Yu, J. Bi, and V. Tresp. Active learning via transductive experimental design. In *Proc. of the 23rd Intl. Conf. on Machine learning*, pages 1081–1088, 2006.

[17] M. Zuluaga, A. Krause, P. Milder, and M. Püschel. "Smart" design space sampling to predict Pareto-optimal solutions. In *Proc. of LCTES*, pages 119–128, 2012.

Table 6: Average ADRS achieved by different DSE methods given HLS budget b. The parenthesized number after `extreme` denotes N_{next} used for selecting next HLS knob-setting. The parenthesized number after `extreme-RT` denotes N_{rted} used in the randomized TED algorithm. For `extreme-RT`, N_{next} is fixed to 59.

Budget b	20	30	40	50	60	70	80	90	100	110	120
Method	Average ADRS (%)										
Training-Set Size = 10											
state-of-the-art [17]	27.37	20.53	15.39	9.36	5.50	2.88	1.56	1.11	0.84	0.20	0.00
basic	21.54	14.95	7.13	3.85	2.23	1.45	1.05	0.50	0.17	0.02	0.00
basic-ST	165.79	98.43	34.63	11.37	6.88	3.82	2.13	0.93	0.24	0.01	0.00
extreme (59)	21.09	16.52	9.49	4.71	3.27	2.54	2.17	1.88	1.49	0.39	0.08
extreme (79)	22.02	13.11	6.79	5.22	3.41	2.70	2.30	1.97	1.27	0.36	0.07
extreme (99)	25.13	18.48	9.51	5.48	3.45	2.42	2.12	1.83	1.46	0.45	0.07
extreme-RT (59)	25.20	18.65	10.08	5.55	3.51	2.24	1.68	1.37	0.82	0.15	0.03
extreme-RT (79)	16.10	9.69	7.74	4.54	3.14	2.63	2.17	1.62	0.80	0.13	0.02
extreme-RT (99)	19.12	11.77	8.15	4.98	3.23	2.20	1.72	1.40	1.08	0.26	0.06
Training-Set Size = 20											
state-of-the-art [17]	NA	12.95	9.98	8.23	6.90	4.38	2.16	0.59	0.24	0.15	0.15
basic	NA	9.01	3.58	1.74	0.99	0.50	0.34	0.18	0.07	0.03	0.00
basic-ST	NA	15.25	12.39	7.47	3.23	0.97	0.31	0.01	0.00	0.00	0.00
extreme (59)	NA	10.31	6.57	4.02	2.49	1.64	1.22	0.94	0.70	0.24	0.04
extreme (79)	NA	11.50	6.83	4.23	2.25	1.47	1.12	0.95	0.68	0.30	0.09
extreme (99)	NA	11.01	7.20	4.82	2.82	1.86	1.55	1.41	0.97	0.37	0.12
extreme-RT (59)	NA	10.97	7.86	5.28	3.15	2.14	1.77	1.41	0.85	0.26	0.04
extreme-RT (79)	NA	9.45	6.09	3.71	1.56	0.95	0.75	0.70	0.64	0.25	0.03
extreme-RT (99)	NA	9.23	5.82	3.52	2.00	1.49	1.34	1.30	0.64	0.25	0.02
Training-Set Size = 30											
state-of-the-art [17]	NA	NA	10.43	6.98	4.06	2.88	1.88	1.10	0.28	0.15	0.03
basic	NA	NA	5.01	2.24	1.05	0.48	0.32	0.16	0.08	0.04	0.01
basic-ST	NA	NA	5.62	2.30	0.98	0.21	0.11	0.01	0.00	0.00	0.00
extreme (59)	NA	NA	7.20	3.72	2.13	1.09	0.64	0.48	0.27	0.21	0.07
extreme (79)	NA	NA	6.46	3.92	2.01	1.20	0.68	0.48	0.25	0.18	0.08
extreme (99)	NA	NA	7.58	4.55	2.23	1.35	0.72	0.39	0.23	0.17	0.15
extreme-RT (59)	NA	NA	8.15	4.30	1.96	1.05	0.65	0.51	0.40	0.21	0.03
extreme-RT (79)	NA	NA	6.95	4.21	2.36	1.31	0.84	0.70	0.64	0.41	0.20
extreme-RT (99)	NA	NA	7.02	3.89	2.26	1.32	0.91	0.62	0.45	0.39	0.03
Training-Set Size = 40											
state-of-the-art [17]	NA	NA	NA	8.16	5.33	3.02	1.43	1.28	1.07	0.33	0.18
basic	NA	NA	NA	4.81	1.86	1.07	0.48	0.27	0.10	0.03	0.01
basic-ST	NA	NA	NA	3.12	1.11	0.29	0.17	0.16	0.00	0.00	0.00
extreme (59)	NA	NA	NA	5.37	2.64	1.32	0.58	0.17	0.12	0.05	0.01
extreme (79)	NA	NA	NA	4.86	2.58	1.33	0.78	0.35	0.15	0.05	0.01
extreme (99)	NA	NA	NA	5.53	2.52	1.12	0.62	0.28	0.10	0.06	0.01
extreme-RT (59)	NA	NA	NA	5.50	2.85	1.78	1.09	0.52	0.40	0.24	0.05
extreme-RT (79)	NA	NA	NA	5.70	2.95	1.64	1.04	0.53	0.42	0.27	0.06
extreme-RT (99)	NA	NA	NA	4.90	2.95	1.78	0.96	0.19	0.10	0.04	0.01
Training-Set Size = 50											
state-of-the-art [17]	NA	NA	NA	NA	6.33	3.75	2.12	0.88	0.63	0.39	0.30
basic	NA	NA	NA	NA	3.33	1.31	0.69	0.42	0.25	0.09	0.03
basic-ST	NA	NA	NA	NA	2.22	0.56	0.29	0.19	0.14	0.00	0.00
extreme (59)	NA	NA	NA	NA	2.71	1.30	0.90	0.43	0.19	0.11	0.03
extreme (79)	NA	NA	NA	NA	3.36	1.64	0.89	0.43	0.17	0.09	0.03
extreme (99)	NA	NA	NA	NA	3.18	1.62	0.87	0.46	0.17	0.08	0.03
extreme-RT (59)	NA	NA	NA	NA	3.65	1.97	1.22	0.87	0.55	0.42	0.26
extreme-RT (79)	NA	NA	NA	NA	3.19	1.49	0.82	0.39	0.16	0.07	0.04
extreme-RT (99)	NA	NA	NA	NA	2.40	1.09	0.74	0.56	0.34	0.27	0.19
Training-Set Size = 60											
state-of-the-art [17]	NA	NA	NA	NA	NA	4.84	2.47	1.38	0.71	0.46	0.23
basic	NA	NA	NA	NA	NA	2.77	0.85	0.55	0.38	0.20	0.09
basic-ST	NA	NA	NA	NA	NA	2.38	0.79	0.32	0.19	0.09	0.00
extreme (59)	NA	NA	NA	NA	NA	2.41	1.21	0.73	0.41	0.18	0.09
extreme (79)	NA	NA	NA	NA	NA	2.62	1.16	0.66	0.37	0.13	0.07
extreme (99)	NA	NA	NA	NA	NA	2.19	1.10	0.65	0.39	0.14	0.08
extreme-RT (59)	NA	NA	NA	NA	NA	1.97	0.90	0.61	0.39	0.14	0.05
extreme-RT (79)	NA	NA	NA	NA	NA	1.93	0.91	0.62	0.39	0.11	0.06
extreme-RT (99)	NA	NA	NA	NA	NA	1.78	0.94	0.61	0.37	0.14	0.07