A No-Human-in-the-Loop Methodology Toward Optimal Utilization of EDA Tools and Flows

Andrew B. Kahng†‡, Shiriram Kumar‡ and Tushar Shah†
†CSE and ‡ECE Departments, University of California, San Diego
abk@ucsd.edu, sckumar@ucsd.edu, tfshah@ucsd.edu

Abstract—Designers access to leading-edge IC technology is challenged by the difficulty and cost of today’s design process. Both human cost (i.e., engineering expertise and effort) and schedule cost (i.e., design schedule) are barriers to design at the leading edge. Indeed, the recently-announced DARPA IDEA program aims to reduce human cost for RTL-to-GDSII layout implementation to “no-human-in-the-loop” levels, and schedule cost to “24-hour turnaround time” levels. In advanced nodes, an added challenge to cost-effective design is that tools and flows are increasingly complex and noisy. Notably, very minute perturbations to constraints can result in very large variations in design flow outcomes. Yet, product companies must achieve the best possible design quality, within prescribed bounds of humans, licenses, CPUs, schedule and with acceptable probability of success. This paper presents a new methodology to underlie future no-human-in-the-loop RTL-to-GDSII implementation. We first document the impact of tool noise in the synthesis, place and route (SP&R) flow on final area and power of the routed and optimized design. We cast the problem of optimally applying a budget of tool runs and schedule across different synthesis target frequencies as an instance of the classic multi-armed bandit problem, and implement a no-human-in-the-loop methodology that strategically samples target frequencies of individual tool runs. We also address a second, “profit maximization” formulation with the multi-armed bandit framework. Our sampling approaches show significant improvements over naive approaches, and the potential to improve design quality within prescribed resource and schedule limits.

I. INTRODUCTION AND MOTIVATION

IC design in advanced nodes requires substantial effort, cost and time. Thus, a design cost barrier blocks system designers’ access to advanced technology. Human engineering costs are a major component of design cost, as documented in the ITRS Design Cost Model [1] [27] [33]. A shortage of human expertise can also block the ability to successfully design in advanced nodes. Recent initiatives such as the DARPA CRAFT (Circuit Realization at Faster Timescales) [3] and IDEA (Intelligent Design of Electronic Assets) [4] programs highlights the criticality of the design cost challenge. Notably, IDEA seeks “the capability for a “no human in the loop,” 24-hour design framework that would enable even nonexperts to design complex electronic technologies”. Both of the targeted reductions – human effort, and design schedule – are extremely challenging.

IC design also becomes more difficult with growing unpredictability of design tools and flows. With increased design complexities, EDA tools incorporate ever more sophisticated algorithmic and optimization techniques at all stages of the design flow. Since many underlying problems (mapping, partitioning, routing, etc.) are NP-hard, additional heuristics are used to solve these problems within practically useful runtimes. This results in unpredictability, or “noisiness”, of tool outcomes, particularly at the limits of what the tool can achieve (see below). Here, noisiness refers to large variations in tool outcomes when designer-specified constraints are slightly perturbed. This unpredictability causes iterations in the design process and exacerbates the design effort challenge.

In this work, we address the goal of “no human in the loop” while explicitly accounting for the tool/flow noisiness which makes design outcomes unpredictable. We develop a model for observed noise in design tools and propose the use of multi-armed bandit (MAB) sampling strategies by which this noise can be learned and exploited. This results in a completely automated, no-human-in-the-loop design methodology – toward obtaining best possible design outcomes, within a given design resource consisting of tool licenses and design schedule. We confirm through commercial EDA tool runs in 14nm and 28nm commercial environments, as well as through simulations, that our no-humans MAB approach outperforms simple “denoising” runs that are typically applied (by human engineers). We apply the MAB approach to two problems. First, we address the problem of maximizing performance (i.e., operating frequency) subject to bounds on area, total power, and total number of tool runs made. Second, we address the problem of maximizing “profit” (where “profit” is monotone in the ratio of frequency/area, and can be viewed as the opposite of “cost”), again given a budget of tool runs. We show that each of these can be formulated as MAB problems, with different reward functions.

Motivating Studies: We have performed extensive experiments that confirm the need to comprehend “noisy” behavior of EDA tools when attempting to maximize quality of design outcomes. In the following, we summarize outcomes of synthesis, placement and routing (SP&R) experiments on three testcases in two technologies: (1) the AES encryption core obtained from OpenCores [30] and implemented in a foundry 28nm FDSOI technology, (2) an Arm CortexM0 core implemented in a foundry 14nm FinFET technology, and (3) and an Arm CortexM3 core implemented in a foundry 14nm FinFET technology. To avoid exposing proprietary information of the IP owner, we scale reported values of M0 area, M0 frequency, M3 area and M3 frequency by fixed multiplicative factors that are each between 0.80 and 1.25. These four multiplicative factors are used consistently throughout this paper, and do not change any of the methods or conclusions presented.

We use Synopsys Design Compiler (version L-2016.03-SP4-1) for logic synthesis and Cadence Innovus (version 16.2) for P&R and optimization. These tool versions have in-built features that promote “stable” solution quality. Key tool options set in our flows are summarized in Table I. Importantly, none of our EDA tool-related observations below should be interpreted as benchmarking of any kind.

<table>
<thead>
<tr>
<th>Synthesis Options</th>
<th>P&amp;R&amp;Opt Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLM = 0</td>
<td>Utilization = 0.7</td>
</tr>
<tr>
<td>Compile Effort = high</td>
<td>Aspect Ratio = 0.7</td>
</tr>
<tr>
<td>max area = 0</td>
<td>leakage opt = true</td>
</tr>
<tr>
<td>Flatten All = true</td>
<td>place_opt_design = true</td>
</tr>
<tr>
<td>SI driven route = true</td>
<td>Timing driven route = true</td>
</tr>
</tbody>
</table>

TABLE I: Tool options set in our SP&R flow.

For this testcase, we execute SP&R and optimization flow with various target frequencies. For each flow stage (i.e., S, P or R) and for each target frequency, we perturb timing constraints slightly, corresponding to frequency steps of 2MHz, so that 11 distinct results are obtained. As a specific example, during synthesis, after a target frequency of 2.22GHz, the synthesis clock frequencies are varied from 2.21GHz to 2.23GHz in steps of 2MHz. Similarly, during placement, for a gate-level netlist fixed, we perturb the the target frequency slightly such that 11 distinct tool outcomes are obtained. We also study the overall effect of perturbations in the synthesis timing constraints on the routed and optimized design. We include optimization in all executions of placement and of routing. Figure I shows results of our experiments. In the figure, “target frequency” is the frequency (i.e., clock period) constraint for a particular flow stage, while “synthesis frequency” is a frequency constraint applied specifically during synthesis.

Figure 1 reveals considerable noise at every flow stage. The CortexM0 data (Figure 1 (top)) show a large variation in post-synthesis area when the synthesis target frequency is perturbed only slightly (up to 19.2% area variation across the 11 runs at 770MHz). The variation gradually reduces post-placement (up to 11% at 1.5GHz) and post-route (up to 1.9%)

1To mitigate noisiness of tool or flow outcomes, design teams will run a given tool/flow multiple times with small perturbations to constraints or other initial conditions, and take the best of the resulting solutions forward in the design process.

In this section, we refer to this as “denoising”.

1This is different from non-determinism, wherein different outcomes are observed for exactly the same input parameters.
at 1.8GHz. We see an overall area variation of up to 8% at 1.8GHz in the post-routed and optimized netlist, when the synthesis target frequency constraints are varied in steps of 2MHz. For AES, we observe area variation of up to 8% at 2.4GHz post-synthesis, up to 6% at 2.4GHz post-placement, and up to 2.5% at 2.65GHz post-route; overall variation is up to 9.93% at 2.65GHz (Figure 1 (bottom)). Similar experiments with the 14nm CortexM3 show overall variation in area of up to 6.94% at synthesis frequency of 1.075GHz, and 1.2% at 600MHz. (This noise in today’s design enablement is similar to what [2] [6] observe.) Note that each of the above frequencies is the maximum frequency at which a particular flow stage just meets the timing constraints for that stage. This implies that, in practice, the maximum variation in outcomes occurs at the maximum achievable frequency. I.e., our data suggest that an EDA tool can have larger variations in outcomes when it must “try hard” to meet designer-specified constraints – and this variation reduces if design constraints are relaxed. This phenomenon is apparently well-known to chip implementation teams, but systematically characterizing and exploiting this is still an open challenge [10].

**Contributions and organization of this paper:** Our main contributions are summarized as follows.

- We compile data to inform explicit modeling of EDA tool noise.
- We define the problem of optimal sampling in the presence of noise. We focus on two formulations – (i) maximizing the frequency at which a valid design solution (i.e., sample) is obtained, and (ii) maximizing profitability of a design – given bounds on tool license and design schedule resources.
- We cast (i) and (ii) as instances of the multi-armed bandit (MAB) problem and propose an effective sampling strategy for each.
- We perform experiments to quantify relative merits of the MAB sampling strategies with both real and synthetic datasets.
- We develop an in-house tool to perform no-human-in-the-loop optimal sampling using the MAB framework. Results indicate that our proposed approaches can result in up to 12% improvement in profit and performance as compared to naive sampling strategies.

Overall, our work suggests that improved awareness, modeling and exploitation of tool and flow noise can improve IC design outcomes within schedule and resource constraints, and consistent with future “no-human-in-the-loop” design methodologies. In the following, Section II reviews previous studies of noise in EDA tools, and the multi-armed bandit problem. Section III provides a more detailed introduction to the MAB problem and the sampling algorithm that we use to solve it in IC design project scenarios. In Section IV, we formulate and solve in the MAB framework the maximization of expected design quality, and the maximization of chip “profit”, subject to resource and performance constraints. Last, Section V presents our no-human-in-the-loop tool built on the MAB framework, and experimental results with both real and synthetic datasets.

## II. RELATED WORK

The work of Kahng and Mantik [2] studies noise in EDA tools, and documents noise in the output quality of place-and-route tools when semantics-preserving perturbations are made to tool inputs. (That is, the tool outcomes change when input netlists are varied such that functional equivalence is preserved, but instance names, ordering of input data files, etc. are changed.) The work of [5] proposes a Gaussian Mixture model for this kind of tool noise. Noise in EDA flows – specifically, high sensitivity of tool outcomes to perturbations in design-specific constraints – is studied in [6], which observes that minute changes in synthesis timing constraints, clock uncertainty, block aspect ratio, or placement utilization could result in significant differences in the area of the final netlist after place-and-route. As an example, total instance area is observed to vary by 16.4% when the placement utilization is changed by 1%. The work of [7] studies noise in the context of incremental and full optimization at various stages of IC physical implementation. [7] identifies design flow stages for which solution quality under incremental optimization is more robust – i.e., exhibiting less noise – when processing incremental Engineering Change Order (ECO) changes to the design. Conversely, other flow stages show unstable – i.e., more noisy – solution quality under incremental design changes, and are hence more suited to full optimization. While these works demonstrate the existence of noise in EDA tools and flows, they do not show any systematic way to exploit such behavior.

To the best of our knowledge, no previous work studies the allocation of “noisy” tool runs in a resource-constrained context. A key contribution of our present work is to cast this as a form of the classic MAB problem. The MAB problem has been studied extensively in the past, but its application in the EDA is extremely sparse. Previous application of the MAB framework in EDA can be found in [9]. In this work, the authors use the MAB algorithm to optimally explore and exploit the design space of parameters for FPGA compilation tuning. However, [9] does not consider any notion of noise in tools, and considers each combination of parameters to be a single outcome, as compared to a distribution. The authors use a standard Upper Confidence Bounds (UCB) reward model [13] that is suitable for maximizing the mean reward. By contrast, in our present work, we focus on developing specialized reward models to learn more efficiently about the parameter space and optimize the maximum reward (rather than the mean reward) obtained within a given budget of runs. The MAB framework has also found application in software benchmark tuning, as studied by Ansel et al. in [8]. The exploration-exploitation tradeoff encountered in sequential sampling from independent Gaussians, which forms the basis for the MAB framework, was initially studied by Robbins [11]. Since then, many theoretical approaches to this problem have been proposed([12] gives a comprehensive overview), and the algorithms in [13] have been shown to work well in practice.

### III. THE MULTI-ARMED BANDIT PROBLEM AND STRATEGIES

We now introduce the multi-armed bandit (MAB) problem and briefly describe algorithms that have been shown to work well in practice. In...
particular, we focus on the Thompson Sampling (TS) algorithm.

**Overview of the MAB Problem.** Consider the following scenario. We are given a slot machine with $N$ arms, each arm having an unknown distribution of rewards. Given $T$ pulls (also referred to as plays or samples) on arms of the slot machine (more precisely, $T$ iterations), we would like to maximize the total reward that we obtain. The reward obtained from a given arm is i.i.d. (independent, and identically distributed). The goal is to maximize the expected total reward $E\left[\sum_{t=1}^{T} a_t\right]$, where $a_t$ is the arm played at iteration $t$. This involves an explore-exploit tradeoff where we draw samples to learn the parameters of the distributions while simultaneously maximizing our rewards. An alternate, equivalent form, often seen in the literature (e.g., [11]) is regret minimization, where regret can be described as the amount lost due to not playing the optimal arm in each step; the total regret is the sum of regrets over all steps. Let $r^*$ be the reward for the optimal arm at any step $j$. Then, the regret for that step is $r^* - r_i$ and the expected total regret is $E\left[\sum_{j=1}^{T} r^* - r_j\right]$. Algorithms that solve the multi-armed bandit problem provide guarantees on regret. The Thompson Sampling (TS) algorithm used in this paper has a logarithmic bound on regret as a function of $T$.

**Algorithms for the MAB Problem.** Many algorithms have been proposed to solve the MAB problem in various settings. Our preliminary studies have considered a variety of commonly-used MAB algorithms, including softmax sampling, ε-Greedy sampling and Thompson Sampling (TS) [16][15][18][19][14]. Extensive empirical comparisons show that TS is more robust in our design tool/flow sampling context, across a wide range of settings, compared to other algorithms. Thus, in the remainder of this work we focus on TS, which we now describe.

**Thompson Sampling Algorithm.** The Thompson Sampling (TS) algorithm dictates that we distribute samples among arms in proportion to their probability of performing the best. To accomplish this, we start with a prior distribution on the parameters of each arm, and update it as we obtain new samples to form the posterior distributions. At each iteration, parameters are sampled from the posteriors and used to calculate the rewards. The arm with the highest reward value is then played. The choice of priors is problem-specific, and we discuss this along with our two IC design-motivated problem formulations in Section IV. The procedures for inferring the posteriors and calculating the expected rewards are discussed in Section IV.

**IV. MAB Formulations to EXPLOIT EDA TOOL NOISE**

In this section, we first establish notation used in the remainder of the paper. We also formulate two problems that optimize metrics of interest to IC design teams as instances of the multi-armed bandit problem. We discuss details of how the reward distributions are modeled, such as the prior distributions we assume, and the calculation of the posteriors. Methodology to calculate the expected gains for each problem is also mentioned briefly.

**A. Notation and assumptions**

Our discussion below uses the following notation and assumptions.

- $A_{\text{max}}$ is the maximum area threshold.
- $S_{\text{min}}$ is the minimum slack threshold.
- $N$ is the total budget of available tool runs.
- $k$ is the number of parallel tool runs conducted in each iteration. (This is limited by the number of licenses and compute servers available.)
- $\{f_1, \ldots, f_p\}$ are $p$ discrete target frequencies, each representing an arm of the MAB.
- $T$ is the maximum number of iterations, with each iteration corresponding to $k$ parallel tool runs. (This reflects a chip design project’s schedule limit, e.g., a 60-day schedule would correspond to $T = 20$ if three days are needed to run SP&R.)
- $a_t$, $s_t$ are the area and slack values obtained at iteration $t$.
- $h_p$ is the probability of obtaining a sample (tool outcome) that satisfies both slack and area constraints at arm $p$.
- We consider three information regimes according to the amount of prior information we have about the distributions. The unknown regime is when we have no prior knowledge. The partially known regime is when we have some prior knowledge, e.g., that certain metrics of interest are normally distributed. The complete information regime is when we know a priori the distributions of all metrics of interest for each arm of the MAB.

- We also consider three sampling strategies that reflect the tradeoff between schedule reduction and gaining feedback from previous tool runs. Batch sampling schedules all tool runs at once with no feedback, i.e., $T = 1$, $k = N$. Sequential sampling schedules one tool run at a time so as to allow for maximum feedback, i.e., $T = N$, $k = 1$. Hybrid sampling is an intermediate strategy that schedule runs in ”mini-batches” of size $k$, with one mini-batch scheduled per iteration, i.e., $N = Tk$. After each mini-batch, we collect feedback that informs subsequent iterations.

In practical industry settings, IC designers must supply constraints to EDA tools that optimize metric(s) of interest, subject to upper bounds on turnaround time ($T$) and number of available tool licenses ($k$). Information about the distribution of the metric is not generally known in advance, or is at best partially known. We consider the problem of finding a synthesis timing constraint that results in a netlist with optimum area (see Problem 1 below) and cost (see problem 2 below), after completion of P&R and optimization. The design project schedule and the maximum number of parallel runs together determine the total budget of tool runs $N$. We focus on the case where $k << N$, i.e., the problem is predominantly sequential and we have zero or limited prior knowledge about the distributions of the metrics. This is similar to a delayed reward situation in a multi-armed bandit problem, where we make $k$ plays before observing the total reward obtained. In Section V, we investigate this assumption further by measuring achieved performance (frequency) as we sweep $k$.

Next, we discuss two problems where we optimize metrics of interest in IC implementation. We will define a reward function for each of the problems and show that maximizing reward will optimize the quantity of interest. We also discuss our assumptions about the reward distributions, priors and posteriors. We note that the runtime of our algorithms ranges from a few seconds to a few minutes in our experiments, and is hence negligible in comparison to the time taken for a tool run. We do not discuss the runtime complexity of our algorithms further.

**B. PROBLEM 1: MAXIMIZING PERFORMANCE**

In Problem 1, we aim to maximize performance of a design, subject to area and timing constraints, having no prior knowledge of achievable design metrics (area, power and maximum operating frequency). Typically, designers try to meet a specified target frequency of operation (spec) for a design. When this spec is not met, multiple iterations of Engineering Change Orders (ECOs) are performed until the target frequency is met. This process consumes considerable human effort and schedule. Here, we seek to characterize and exploit tool noise with a strategy that increases the probability of meeting a given frequency spec (as compared to naive denoising), within a given budget of tool runs.

**Problem 1 Statement:** Given an upper bound on area $A_{\text{max}}$, and a worst-case slack threshold (lower bound) $S_{\text{min}}$, distribute a budget of $N$ tool runs across $p$ discrete target frequencies $\{f_1, \ldots, f_p\}$ so as to maximize the frequency $f^*$ at which there is at least one sample with area $A < A_{\text{max}}$ and $S > S_{\text{min}}$.

**Formulation:** Let the boolean random variable $g_i$ be 1 if the $i$th frequency is the maximum frequency at which we have a sample that meets the slack and area constraints. Our problem can be written as $\max_{n_i} n_i E[g_i]$. Let $f_{\text{max,i}}$ be the frequency maximum having at least one point that satisfies the constraints at step $t$. Consider the reward function

$$r_i = \begin{cases} f_{t} - f_{\text{max},t-1} & a_t < A_{\text{max}} \text{ and } s_t > S_{\text{min}} \\ 0 & f_t < f_{\text{max},t-1} \text{ or } a_t > A_{\text{max}} \text{ or } s_t < S_{\text{min}} \end{cases}$$

where $f_t$ is the frequency at which the sample is drawn in iteration $t$. If $f_{\text{max,N}} = f_{\text{max},t} + r_t$, we see that $f_{\text{max,N}}$ is the maximum frequency where we have a sample that meets the slack and area thresholds. Thus, $f_{\text{max,i}} = \sum_{t=1}^{N} r_t$, and we would like to solve

$$\max_{n_i} E[f_{\text{max,i}}] = \max_{n_i} \sum_{t=1}^{N} E[r_t]$$

where $n_i$ represents the number of samples drawn at arm $i$. This reward maximization problem can be solved by MAB algorithms.

**Modeling Rewards:** We use a Bayesian approach to model the probability of success at each frequency. A natural conjugate prior that is widely
used for binomial random variables is the beta prior. As we obtain samples, we update the parameters of the prior and obtain a posterior distribution on the probability of success for each frequency. As the frequency increases, the area increases and we hit the wall of slack. Having observed this in multiple designs, we incorporate this into the reward model by truncating the posterior to the region $h_0 < h_1 < \cdots < h_p$.

The expected value of the reward function, as we can see from Equation 1, depends only on the expected probability of success. This is simply the mean of the posterior distribution. We estimate this value using samples drawn from the posterior using Markov Chain Monte Carlo (MCMC) methods [32]. The Thompson Sampling algorithm uses the samples from the posterior joint distribution to calculate the reward for each arm. The arm with the highest reward value is sampled from next (this means that each arm is picked according to the probability that it performs the best).

C. PROBLEM 2: MAXIMIZING PROFIT

Problem 2 seeks to maximize the “profit” of a chip design, by optimizing a function of both its area and its frequency. Manufacturing cost of a chip increases with area. On the other hand, the selling price of a chip generally increases with operating frequency. For purposes of our study here, we define a metric, “cost”, as a measure of the combined effect of area and frequency on the profitability of a chip. Our aim is to minimize cost, thus maximizing profitability of the chip.

Problem Statement: Given an area threshold $A_{\text{min}}$, a slack threshold $S_{\text{min}}$, and a budget of total tool runs $N$, distribute the tool runs across the frequencies $\{f_1, \ldots, f_p\}$, so as to minimize the expected minimum cost obtained.

Formulation: Let $a_i$ be the area obtained in the $i^{th}$ tool run and $f_i$ be the frequency at which the tool run was conducted. Then $c_i = a_i / f_i$ is the associated cost. The cost minimization problem can then be stated as

$$\min_{n_1, \ldots, n_i} \mathbb{E}[c^*]$$

where $c^* = \min_i \{c_i | a_i < A_{\text{min}} \text{ and } s_i > S_{\text{min}}\}$

where $n_i$ represents the number of samples drawn at arm $i$. Next, consider the reward function.

$$r_i = \begin{cases} 
  c_{i-1} - x & x < c_{i-1} \\
  0 & x \geq c_{i-1} 
\end{cases}$$

where $x$ is the sample cost obtained in the $i^{th}$ run. If we use the recursive relation $c_i = c_{i-1} - r_i$, we see that

$$c_i = \begin{cases} 
  c_{i-1} & x \geq c_{i-1} \\
  x & x < c_{i-1} 
\end{cases}$$

This means that $c_i = \min_{1 \leq j \leq i} c_j$. Also, we know that $c_i = c_0 - \sum_{j=1}^{i} r_j$. Using expectations, we get $\mathbb{E}[\min_{1 \leq j \leq i} x_j] = c_0 - \sum_{j=1}^{i} \mathbb{E}[r_i]$. Thus, our problem can be reformulated as maximizing the sum of expected rewards over the total number of tool runs that can be made within the given budget. Note that this problem has a natural dimensioning returns characteristic. Given the normal distribution, the expected minimum value saturates as samples sizes become very large. Thus we expect to see significant improvements in performance over naive algorithms for small budget sizes. We explore this further in our experiments in Section V.

Modeling Rewards: We hypothesize that the rewards for each arm $p$ are normally distributed with some mean $\mu_p$ and standard deviation $\sigma_p$ (recall Figure 3) and confirm this using a standard hypothesis test [35].

We choose the following priors for the mean and standard deviations: $\sigma_p \sim \Gamma(\alpha_p, \beta_p)$ and $\mu_p \sim N(\mu_0, \lambda K_0)$. The model is shown in Figure 2. The Gamma and Normal distributions are natural priors for the mean and standard deviation of a normal random variable. However, our choice of $K_0$ warrants some additional discussion. The $K_0$ value is chosen to reflect our belief that arms that are closer together will have means that are similar. This enforces a smoothness in the values of means across arms and helps our algorithm quickly identify promising regions for further exploration/exploitation.

Parameter samples are generated from the posterior using the No-U-turn sampler [26] which is a Hamiltonian Monte Carlo [29] method. Since the expected reward is a deterministic function of these parameters, it can be calculated by averaging across these samples. TS uses the reward values calculated from the samples to pick the best arm and play it next.

V. EXPERIMENTAL SETUP

In this section, we experimentally study the relative benefits and drawbacks of our proposed MAB sampling strategies, in various potential design project scenarios. In particular, we seek to understand how our strategies perform in the cases when: (1) the design team has very few vs. very many available tool runs, i.e., small vs. large values of $k$ and/or $N$; (2) the design schedule is limited versus relaxed, i.e., small versus large values of $T$; and (3) meeting the design’s area or performance specification is “difficult” (i.e., samples from available MAB arms will only rarely satisfy both area and timing constraints) versus “easy” (i.e., samples from available MAB arms frequently satisfy both constraints);

A No-Human-in-the-Loop Tool. We have developed a fully automated, “No-Human-in-the-Loop” (NHIL) tool in Python/Perl/Tcl that defines and schedules runs using the proposed MAB algorithms. We use this tool to run our algorithms on our three designs. To provide further insight, we also study the design scenarios of Section V using a simulator based on (thousands of collected SP&K runs of) denoising data for these designs. This enables us to quantify benefits of MAB over denoising, within available compute resources. More specifically, the execution of a MAB strategy on a particular design project scenario requires us to capture several basic parameters. To specify a design project scenario, we require models of $[\mu, \sigma]$ of (area, slack) as functions of target frequency, and we require the $p$ target frequencies. To specify a particular design resource or design objective, we must know the acceptable thresholds of chip area $A_{\text{max}}$ and of timing slack $S_{\text{min}}$, the available numbers of tool runs $N$ and the available licenses $k$. We obtain the $\mu$ and $\sigma$ values from denoising experiments on the 28nm AES, 14nm CortexM0 and 14nm CortexM3 designs.

A. Design of Experiments

We now describe our design of experiments to address the questions above. We consider two practical problems: (i) maximizing frequency, given an area threshold $A_{\text{max}}$, slack threshold $S_{\text{min}}$ and budget of available tool runs $N$ and (ii) maximizing profit, given an area threshold $A_{\text{max}}$, slack
threshold $S_{\text{min}}$ and a budget of tool runs $N$. As mentioned above, we obtain the $p$ and $\sigma$ values from denoising experiments.

For each of these problems, we perform the sets of experiments listed below, and compare the performance of the MAB algorithms against denoising. In Section V-B, we compare trends seen in the simulated model to those seen in practice on real designs.

- **Experiment 1: Effect of design schedule constraint.** We test the effect of different design schedule constraints on our algorithms by using the simulation setup described previously. We fix $N = 200$ and vary the number of parallel runs $k$, in each iteration. $k = \{5, 10, 20, 40\}$. We compare our simulated results against those obtained from actual tool runs on CortexM0 using the NHIL tool described above.

- **Experiment 2: Effect of number of available tool runs.** We test the impact of the number of tool runs available on the performance of our algorithms. $N = \{50, 100, 150, 500\}$, keeping $k = 1$, i.e., the purely sequential scenario.

- **Experiment 3: Effect of area constraints [Not applicable for Problem 2].** We test the performance of our algorithms in different scenarios of varying difficulty to meet designer-specified constraints (in this case maximum area constraints). We target three scenarios: (i) when it is easy to meet constraints, (ii) when only certain frequencies can meet constraints, and (iii) when it is very difficult to meet constraints.

In summary, three experiments are performed for Problem 1 (Experiment 1, Experiment 2, Experiment 3), and two experiments are performed for Problem 2 (Experiment 1 and Experiment 2). For each experiment, we compare the outcomes of our algorithm against denoising. From this point on we use “denoising” to refer to the process of uniformly spreading a budget of tool runs $N$ across $p$ arms.

**B. Experimental Results: Problem 1**

**Experiment 1: Effect of design schedule constraint.** In this set of experiments, we sweep the number of parallel runs, $k$, in an iteration: $k = \{5, 10, 20, 40\}$ while keeping $N$ constant at 200. The goal of these experiments is to study the behavior of our algorithm when runs are not executed in a completely sequential fashion. The results of these experiments are shown in Figure 5b. We compare the outcomes of our simulation setup versus the outcomes of actual tool runs by using the tool described above on a CortexM0 core implemented in 14nm FinFET technology. For our tool run experiments, we set our area constraints $A_{\text{max}} = 4100$. A comparison of sampling patterns at each iteration for $k = 5$ and $k = 10$ is shown in Figure 4.

In the most sequential scenario, i.e., $k = 5$, we obtain a valid sample at 2.001GHz. This is significantly higher than more parallel scenarios, $k = 10, 20, 40$, wherein the highest frequency at which a valid sample was obtained was observed to be $\sim 1.8$GHz. We confirm this behavior through 50 simulations, as shown in Figure 5b. We observe a 60% probability of obtaining a maximum frequency of 2GHz, when $k = 5$. In comparison, when $k = 10, 20, 40$, the probability of obtaining a valid sample at 2GHz is less than 30%. For denoising, the probability of obtaining a valid sample at 1.9GHz, or greater, is 36%. This is significantly lower than what our algorithm achieves for all values of $k$.

**Experiment 2: Effect of number of available tool runs.** We perform five experiments here with $N = \{50, 100, 150, 200, 500\}$ and $k = 1$ for AES, CortexM0 and CortexM3. Quantitatively similar trends are observed for all designs. We show the results of experiments on CortexM0 in Figure 5a. As the value of $N$ increases from 50 to 500, the probability of obtaining a valid sample at 2GHz increases from $< 20\%$ to $> 50\%$. Probability of obtaining samples at 2.1GHz and higher is very small for all values of $N$. In comparison, for denoising, for $N = 100, 150, 200, 1.8$GHz was the maximum frequency at which a valid sample was obtained $\sim 73\%$ of the time. Probability of obtaining a valid sample at 1.9GHz or greater is less than 20%. For $N = 500$, the maximum frequency at which a valid sample is obtained is 1.9GHz 66% of the time. This probability is negligibly small for frequencies $> 1.9$GHz. From our simulations we can conclude that our algorithm performs better than denoising for all values of $N$.

**Experiment 3: Effect of area constraints.** Here we test the effectiveness of our algorithms in scenarios of varying difficulty. We compare the performance of the algorithm when the constraints are easy, medium, and hard to meet. As a specific example, for CortexM0, $A_{\text{max}} = 4000$ would be an easy constraint for 1.8GHz, $A_{\text{max}} = 3750$ would be a challenging constraint. We plot our results in Figure 5c. In the “easy” constraint scenario, as expected, the MAB always achieves a valid sample at 1.8GHz. While in the challenging constraint scenario, it is unable to achieve a valid sample in any case. In the “medium” constraint scenario, the MAB algorithm achieves a valid sample at 1.8GHz 60% of the time. In comparison, for denoising, we achieve a maximum frequency of 1.8GHz only 40% of the time in the easy scenario, 1.8% for the medium scenario. For the challenging scenario, results are comparable between the algorithms. Denoising achieves a maximum frequency of 1.7GHz 94% of the time.

**C. Experimental Results: Problem 2**

We evaluate the performance of our MAB algorithm for Problem 2, using simulated data. Simulation settings described above are used here. Figures 6(a) and 6(b) show the expected gain in percentage over the mean at the optimal arm for denoising and MAB algorithms as a function of budget size for two designs. If we ignore noise, the optimal algorithm would be to sample once at each arm and choose the best from the resulting samples. The expected cost in this case would be the mean of the optimal arm. We measure the expected gain in cost over this setting for both algorithms.

We use a radial basis function (rbf) kernel [34] to generate the prior parameter $r_0$ that controls the smoothness of the mean values across arms. The degree of smoothness is controlled by the scale parameter of the rbf kernel. The prior mean estimate $r_0$ is chosen to be a constant across arms that is set per design, $a$ is set to 25, $b$ to 0.5 and $\lambda$ to 40000. This corresponds to a flexible prior that does not carry much information other than smoothness. This allows the model to adapt quickly to incoming data. Using these settings, we run our no-human-in-the-loop tool on CortexM0. We would need a budget of 12% in the cost (equivalently area) at 2GHz, as compared to the mean. In comparison, denoising is able to achieve a 2.9% improvement in the cost at 2GHz. This is a significant level of improvement, achieved with no human effort and with the same compute resource as naive denoising.

**Experiment 1: Effect of design schedule constraint.** As the value of $k$ increases for our chosen budget size of 100, the sampling distribution can be updated less frequently. This leads to inefficiency and we can see that the performance reduces as $k$ increases for any budget size. This result is similar to what we observe in Problem 1. However, even with $k = 50$, i.e. only two rounds of updates, the MAB algorithm still does significantly better that the denoising algorithm with a budget size of 100 runs.

**Experiment 2: Effect of number of available tool runs.** From Figures 6(a) and 6(b) we see that the percentage performance improvement in the MAB algorithm grows much faster than that of the denoising approach for all values of $k$. As $N$ increases, the percentage improvement saturates for the MAB algorithm. This is due to the diminishing returns nature of the problem where successive improvements become smaller as our solution gets better. Thus, as the value of $N$ increases, the difference in performance between the two algorithms will reduce. However, to achieve the results shown by the MAB algorithm with $k = 5$ in 100 tool runs, denoising would need a budget of approximately 2100 tool runs which represents a significant compute overhead.
Fixed multiplicative scaling of area values, and of frequency values, is applied per IP owner requirements. of varying difficulty of constraints on maximum frequency achieved, showing the probability that 

\[
\text{Probability of Success} \geq 98%
\]

by the third iteration and increases all the way up to 100% for \( N = 50, 100, 150, 200, 500 \) for \( k = 10 \). Fixed multiplicative scaling of area values, and of frequency values, is applied per IP owner requirements.

**Fig. 5:** (a) Effect of total available tool runs on maximum frequency achieved, showing probability that \( f_{\text{max}} = 1.9, 2, 2.1, 2.2 \) for \( N = 50, 100, 150, 200, 500 \), \( k = 10 \). (b) Effect of mini-batch size \( k \) on the maximum frequency achieved, showing probability that \( f_{\text{max}} = 1.9, 2, 2.2 \) for \( \kappa = 5, 10, 20, 40 \), \( N = 200 \). (c) Effect of varying difficulty of constraints on maximum frequency achieved, showing the probability that \( f_{\text{max}} = 1.7, 1.8 \) for \( \kappa_{\text{max}} = 3750, 3860, 4000 \), \( k = 10 \), \( N = 200 \). The smoothness property to achieve a good approximation of the mean costs in just two iterations for all three scenarios.

**Fig. 6:** (a) Percentage (%) improvement in cost over the mean obtained by the denoising and MAB algorithms for several budget sizes and levels of parallelism \( k \) for Arm CortexM0. (b) Results of the same experiment for Arm CortexM3. (c) Sampling rates of the MAB algorithm at each frequency across iterations. Each row corresponds to an iteration and each column corresponds to a frequency. The shade of a cell gives the probability of sampling at a frequency at an iteration (see legend on right).

The results for AES look similar, with MAB showing a 3.8% improvement, and denoising showing a 1% improvement over the mean. We omit the % improvement plot for AES since it looks very similar to CortexM0. We instead show the sampling rates across frequencies and iterations in Figure 6(c). We see that the algorithm is able to identify an optimal sampling region in as few as three iterations for AES core. The sum of the sampling rates for AES at the three best arms reaches 75% by the third iteration and increases all the way up to 98% in the tenth iteration. The algorithm successively eliminates suboptimal regions using the smoothness property to achieve a good approximation of the mean costs in just two iterations for all three scenarios.

**VI. CONCLUSION AND FUTURE WORK**

Our work provides a step toward “no-human-in-the-loop” IC implementation through automated execution of EDA tools/flows in a MAB framework. We confirm that significant noise in EDA tool outcomes still exists in today’s design enablers, and we apply the MAB framework to achieve optimal resource-bounded use of tools in this context. We devise adaptive sampling strategies for two example problem formulations – performance maximization, and profit maximization – applicable under (license, schedule, area, frequency) constraints. In the settings we have tested, our methods can obtain performance and profit improvements of up to 12%, and are never worse, compared to today’s naive sampling approaches such as “denoising”. Thus, our method offers reduction of human effort while improving design outcomes. Furthermore, our proposed MAB approaches inherently provide early estimates of maximum achievable performance for a given design, with a certain confidence level (but, this attribute was not a focus for our work here). This can give early insight into available tradeoffs between design schedule and product performance. Our ongoing work extends the MAB framework to include tool parameter tuning, as well as a richer set of design constraints, in various stages of the RTL-to-GDSII flow.

**REFERENCES**


