BOiLS: Bayesian Optimisation for Logic Synthesis

Antoine Grosnit*
Huawei Noah's Ark Lab
antoine.grosnit@huawei.com

Cedric Malherbe*
Huawei Noah's Ark Lab
cedric.malherbe@huawei.com

Rasul Tutunov
Huawei Noah's Ark Lab
rasul.tutunov@huawei.com

Xingchen Wan
Huawei Noah's Ark Lab
University of Oxford
xingchen.wan@huawei.com

Jun Wang
Huawei Noah's Ark Lab
University College London
w.j@huawei.com

Haitham Bou Ammar
Huawei Noah's Ark Lab
University College London
haitham.ammar@huawei.com

***Abstract*—Optimising the quality-of-results (QoR) of circuits during logic synthesis is a formidable challenge necessitating the exploration of exponentially sized search spaces. While expert-designed operations aid in uncovering effective sequences, the increase in complexity of logic circuits favours automated procedures. To enable efficient and scalable solvers, we propose BOiLS, the first algorithm adapting Bayesian optimisation to navigate the space of synthesis operations. BOiLS requires no human intervention and trades-off exploration versus exploitation through novel Gaussian process kernels and trust-region constrained acquisitions. In a set of experiments on EPFL benchmarks, we demonstrate BOiLS's superior performance compared to state-of-the-art in terms of both sample efficiency and QoR values.**

***Index Terms*—Logic synthesis, Bayesian Optimisation**

I. Introduction

During the pre-mapping stages of logic synthesis (LS), designers uncover a series of structural transformations that improve circuit efficiencies by maximising performance criteria, such as the Quality-of-Results (QoR) [1]. Modern synthesis tools administer those transformations by first representing circuits as And-Inverter Graphs (AIGs) and then employing technology-independent operations to reduce graph sizes while adhering to delay constraints. Although experts devised numerous QoR optimisers [2], [3], exponentially sized exploration spaces, especially in large circuits, still pose formidable challenges to the design of predefined synthesis flows. The quest for scalable and sample efficient solvers has, in turn, stimulated novel research trends that benefit from developments in machine learning (ML) when tailored to LS applications.

Although ML techniques emerged as active areas of research within the electronic design automation pipeline (e.g., in design space reduction [4], placement [5], etc.) , their examination in LS only recently started to gain attention. Prior to this work, the authors in [6] distinguish a handful of ML-inspired approaches based on deep neural networks and reinforcement learning (RL) to obtain optimal structural transformations. For instance, the work in [7] adopts (deep) convolutional neural networks to solve classification problems mapping synthesis flows to QoR levels. Recently [14] also proposes an LSTM-based approach for QoR optimisation, while the authors in [8], [9] extend deep reinforcement learning (DRL) to pre-mapping applications by defining novel Markov decision processes and policies that capture the intricate complexities of LS.

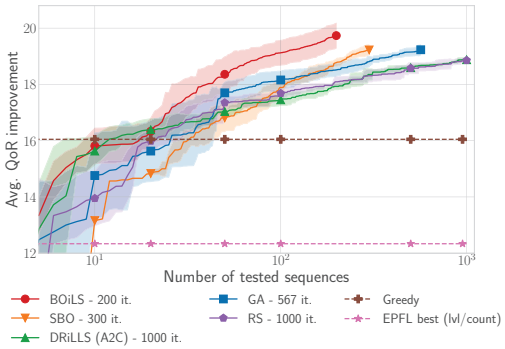

Fig. 1. Average QoR results over 10 EPFL circuits to recover BOiLS' QoR values after only *200* trials.

Albeit their widespread usage, deep learning and RL techniques exhibit *high sample complexities* [10] especially in high-dimensional combinatorial spaces. When applied to LS, such high data demands amount to numerous evaluations within a given circuit, e.g., 10,000 sequences per circuit when adopting convolutional deep networks [7], or over thousands of agent environment interactions in DRL (see Section IV).

We introduce BOiLS, the first Bayesian optimisation solver for LS. BOiLS demands no human intervention and efficiently searches combinatorial spaces, trading-off exploration and exploitation. Our method operates in two steps. First, we fit a surrogate Gaussian process (GP) to QoR data utilising kernels geared towards AIGs transformation sequences. This GP enables both sample efficiency and calibrated uncertainty estimation, which we exploit in the second step to suggest new synthesis flows to evaluate. Here, we harness concepts from local trust-region acquisition function maximisation to effectively handle high-dimensionalities. In a set of experiments on the EPFL benchmark [11], we demonstrate superior QoR performance and better sample efficiencies compared to DRL [8], graph neural network policies [9], genetic algorithms and other search strategies, as well as against the best results from the EPFL leaderboard [11] in 8 out of 10 circuits. We also open-source our code for reproducibility[1].

* Equal contribution.

[1]https://github.com/huawei-noah/HEBO/tree/master/BOiLS

II. Problem Definition

We aim to find an equivalent yet simpler representation of a logic design using a series of primitive transformations. Modern tools [13], [15] express a logic circuit $\mathcal{C}$ as an AIG consisting of two-fanin nodes representing logical conjunction, of input and terminal nodes, and of edges that can be marked to denote logical negation. Our goal is to find a sequence $\texttt{seq} = [\texttt{s}_1, \ldots, \texttt{s}_K] \in \texttt{Alg}^K$ of at most K operations to optimise the AIG structure. Here, $\texttt{Alg} = \{\texttt{A}_1, \ldots, \texttt{A}_n\}$ is a set of n transformation algorithms taken from [15] (e.g., `resub`, `rewrite`,...) that can be executed to alter the AIG. We employ QoR to assess the performance of an evaluated sequence. Having applied `seq` to an AIG, we register both the area (LUT-count), and the delay (Levels) after executing FPGA mapping. Then, we compute the overall effectiveness of `seq`:

$$\text{QoR}_{\mathcal{C}}(\texttt{seq}) = \frac{\text{Area}_{\mathcal{C}}(\texttt{seq})}{\text{Area}_{\mathcal{C}}(\texttt{ref})} + \frac{\text{Delay}_{\mathcal{C}}(\texttt{seq})}{\text{Delay}_{\mathcal{C}}(\texttt{ref})}, \tag{1}$$

where $\text{Area}_{\mathcal{C}}(\texttt{ref})$ and $\text{Delay}_{\mathcal{C}}(\texttt{ref})$ denote area and delay obtained when applying a reference sequence (e.g., `resyn2` [15]). Hence, an optimal sequence $\texttt{seq}^\star$ is given by:

$$\texttt{seq}^\star \in \underset{\texttt{seq}\in\texttt{Alg}^K}{\arg\min}\ \text{QoR}_{\mathcal{C}}(\texttt{seq}) \equiv \underbrace{\arg\max\ \text{-QoR}_{\mathcal{C}}(\texttt{seq})}_{\text{This paper's focus}}. \tag{2}$$

We note two difficulties when seeking $\texttt{seq}^\star$: 1) From Eq. (1), we discern that a closed analytical form of $\text{QoR}_{\mathcal{C}}(\cdot)$ as a function of `seq` is challenging to obtain for any circuit $\mathcal{C}$ since it involves complex algorithmic processes executed over AIGs. In ML, we refer to *functions of unknown analytical* forms as *black-boxes* and seek *efficient data-driven solvers* that optimise for $\texttt{seq}^\star$ based on a handful of sequence evaluations; 2) In attempting a data-driven solution, we remark an exponential growth in the search space $\texttt{Alg}^K$ whose cardinality equates to n^K (11^{20} in our experiments). Due to the black-box nature of $\text{QoR}_{\mathcal{C}}(\cdot)$, it is difficult to assume desirable characteristics like linearity or submodularity that facilitate searching for $\texttt{seq}^\star$.

III. Bayesian Optimisation for Logic Synthesis

A. Bayesian Optimisation (BO) & Gaussian Processes

BO is a gradient-free technique used to optimise expensive black-box functions. BO tackles global optimisation sequentially, selecting at each round t an input probe $\boldsymbol{x}_t$ for evaluation, and acquires a corresponding (noisy) black-box function value $g(\boldsymbol{x}_t) \in \mathbb{R}$. Typically, inputs take on continuous values in bounded domains, whereby $\boldsymbol{x}_t \in \mathcal{X} \subset \mathbb{R}^d$. The goal is to *rapidly* approach the maximum $\boldsymbol{x}^\star = \arg\max_{\boldsymbol{x}\in\mathcal{X}} g(\boldsymbol{x})$ [12], [18], [19]. BO relies on historical data (e.g., $\langle \boldsymbol{x}_1, g(\boldsymbol{x}_1)\rangle, \ldots, \langle \boldsymbol{x}_t, g(\boldsymbol{x}_t)\rangle$ at round t) to *i)* build a surrogate model of the black-box and *ii)* utilise this surrogate to decide on the new input to evaluate. Since both $g(\cdot)$ and $\boldsymbol{x}^\star$ are unknown, learners need to trade off exploitation and exploration during the search process. This is naturally achieved by grounding decisions on the surrogate's *predictive distribution*, where we contrast fully trusting the surrogate's mean prediction or examining unseen inputs. This is accomplished via maximising an acquisition function (see Section III-A2).

1) Gaussian Processes: The first step involves fitting a surrogate model that provides well-calibrated uncertainty estimates. GPs offer a flexible and sample-efficient procedure for placing priors over unknown functions [20]. We can use GPs to directly define distributions over functions, where we write $g(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$. Here, $m(\boldsymbol{x}) = \mathbb{E}[g(\boldsymbol{x})]$ and $k(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}[(g(\boldsymbol{x}) - m(\boldsymbol{x}))(g(\boldsymbol{x}') - m(\boldsymbol{x}'))]$ denote the mean and covariance functions that fully specify a GP. Following [20], we set the mean function to zero, thus having $g(\boldsymbol{x}) \sim \mathcal{GP}(0, k(\boldsymbol{x}, \boldsymbol{x}'))$.

Covariance kernels encode our (smoothness) assumptions about the function $g(\cdot)$ that we wish to learn. Typical kernels $k(\cdot, \cdot)$ is designed such that as two points $\boldsymbol{x}$ and $\boldsymbol{x}'$ get closer, the correlation between $g(\boldsymbol{x})$ and $g(\boldsymbol{x}')$ increases. Given a finite set of input data points $\boldsymbol{x}_{1:n} \equiv \{\boldsymbol{x}_i\}_{i=1}^n$, we can utilise Definition 3.1 to derive the jointly Gaussian prior distribution on the corresponding outputs $\boldsymbol{g} \equiv \{g(\boldsymbol{x}_i)\}_{i=1}^n$: $\boldsymbol{g} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{K}(\boldsymbol{x}_{1:n}, \boldsymbol{x}_{1:n}))$, where $\boldsymbol{K}(\boldsymbol{x}_{1:n}, \boldsymbol{x}_{1:n}) \in \mathbb{R}^{n\times n}$ is the covariance matrix with its $(i, j)^{th}$ entry given by $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

Predictions using GPs: Given training input-output observations $\{\boldsymbol{x}_i, g(\boldsymbol{x}_i)\}_{i=1}^n$, we want to construct the *output predictive distribution* at $\tilde{n}$ test points $\{\tilde{\boldsymbol{x}}_j\}_{j=1}^{\tilde{n}}$. The joint distribution over training and testing function values $\boldsymbol{g}$ and $\tilde{\boldsymbol{g}} \equiv \{g(\tilde{\boldsymbol{x}}_j)\}_{j=1}^{\tilde{n}}$ follows:

$$\begin{bmatrix} \boldsymbol{g} \\ \tilde{\boldsymbol{g}} \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} \boldsymbol{K}(\boldsymbol{x}_{1:n}, \boldsymbol{x}_{1:n}) & \boldsymbol{K}(\boldsymbol{x}_{1:n}, \tilde{\boldsymbol{x}}_{1:\tilde{n}}) \\ \boldsymbol{K}^\mathsf{T}(\boldsymbol{x}_{1:n}, \tilde{\boldsymbol{x}}_{1:\tilde{n}}) & \boldsymbol{K}(\tilde{\boldsymbol{x}}_{1:\tilde{n}}, \tilde{\boldsymbol{x}}_{1:\tilde{n}}) \end{bmatrix}\right),$$

where $\boldsymbol{K}(\boldsymbol{x}_{1:n}, \tilde{\boldsymbol{x}}_{1:\tilde{n}}) \in \mathbb{R}^{n\times\tilde{n}}$ and $\boldsymbol{K}(\tilde{\boldsymbol{x}}_{1:\tilde{n}}, \tilde{\boldsymbol{x}}_{1:\tilde{n}}) \in \mathbb{R}^{\tilde{n}\times\tilde{n}}$ denote the covariance matrices evaluated at all pairs of training and test points and those between test points. Conditioning the multi-variate Gaussian leads to predictive output distributions:

$$\tilde{\boldsymbol{g}}|\boldsymbol{g}, \{\boldsymbol{x}_i\}_{i=1}^n, \{\tilde{\boldsymbol{x}}_j\}_{j=1}^{\tilde{n}} \sim \mathcal{N}(\boldsymbol{\mu}_{\text{post.}}, \boldsymbol{\Sigma}_{\text{post.}}), \tag{3}$$

with posterior mean and covariance given by: $\boldsymbol{\mu}_{\text{post.}} = \boldsymbol{K}(\boldsymbol{x}_{1:\tilde{n}}, \boldsymbol{x}_{1:n})\boldsymbol{K}^{-1}(\boldsymbol{x}_{1:n}, \boldsymbol{x}_{1:n})\boldsymbol{g}$ and $\boldsymbol{\Sigma}_{\text{post.}} = \boldsymbol{K}(\boldsymbol{x}_{1:\tilde{n}}, \boldsymbol{x}_{1:\tilde{n}}) - \boldsymbol{K}(\boldsymbol{x}_{1:\tilde{n}}, \boldsymbol{x}_{1:n})\boldsymbol{K}^{-1}(\boldsymbol{x}_{1:n}, \boldsymbol{x}_{1:n})\boldsymbol{K}(\boldsymbol{x}_{1:n}, \boldsymbol{x}_{1:\tilde{n}})$.

Learning in GPs: The remaining ingredient in a GP pipeline involves introducing the kernel hyperparameters that are tuned using marginals to fit a given dataset best. In standard GPs [20], hyperparameters $\boldsymbol{\theta}$ are determined by minimising the negative log marginal likelihood, leading us to the following:

$$\min_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) = \frac{1}{2}\det(\boldsymbol{K}_{\boldsymbol{\theta}}(\boldsymbol{x}_{1:n}, \boldsymbol{x}_{1:n})) + \frac{1}{2}\boldsymbol{g}^\mathsf{T}\boldsymbol{K}_{\boldsymbol{\theta}}^{-1}(\boldsymbol{x}_{1:n}, \boldsymbol{x}_{1:n})\boldsymbol{g}, \tag{4}$$

where $\det(\cdot)$ is the determinant operator.

2) Acquisition Functions: Proposing novel query points is performed through maximising an acquisition function. In this paper, we adopt the *expected improvement* (EI) [21], which determines new query points by maximising expected gain relative to the best function value observed so far, although other options are possible [18]. At round t of BO, EI is given by $\alpha_{\text{EI}}(\boldsymbol{x}|\mathcal{D}_t) = \mathbb{E}_{\text{GP-predictive}}[\max\{g(\boldsymbol{x}) - g(\boldsymbol{x}_t^+), 0\}]$, where $\boldsymbol{x}_t^+ = \arg\max_{\boldsymbol{x}\in\{\boldsymbol{x}_\ell\}_{\ell=1}^t} g(\boldsymbol{x})$.

B. BOiLS: Bayesian Optimisation for Logic Synthesis

We now introduce BOiLS, a logic-synthesis-specific algorithm that generalises recent works in combinatorial BO [12], [22] for sequential optimisation.

1) GP Kernels for Logic Synthesis: To build a GP surrogate model from QoR data $\mathcal{D}_t = \{\texttt{seq}_i, -\text{QoR}_{\mathcal{C}}(\texttt{seq}_i)\}_{i=1}^{n_t}$ with n_t denoting the number of attempted sequences up-to round t, we assume that $-\text{QoR}_{\mathcal{C}}(\texttt{seq}) \sim \mathcal{GP}(0, k_{\boldsymbol{\theta}}^{(\text{LS})}(\texttt{seq}, \texttt{seq}'))$. Here, kernel's super-script (LS) highlights the need for new logic synthesis functions that measure similarity between categorical sequences of operations applied to AIGs. We represent the sequences as strings of operations, with each character being an algorithm from `Alg`. Similar to [22], [23], we employ the sub-sequence string kernel (SSK) to measure the similarity between strings through the number of common *sub-strings*. Formally, an ℓ^{th} order SSK between strings `seq` and `seq`$'$ is defined as: $k_{\boldsymbol{\theta}}^{(\text{LS})}(\texttt{seq}, \texttt{seq}') = \sum_{\boldsymbol{u} \in \Sigma^\ell} c_{\boldsymbol{u}}(\texttt{seq}) c_{\boldsymbol{u}}(\texttt{seq}')$, where Σ^ℓ denotes the set of all possible ordered collections of up to ℓ characters from our alphabet. Moreover, $c_{\boldsymbol{u}}(\texttt{seq})$ measures the contribution of sub-sequence $\boldsymbol{u}$ to `seq` which is defined using two tunebale hyperameters $\theta_m \in [0,1]$ and $\theta_g \in [0,1]$ that control the relative weighting of long and highly non-contiguous sub-strings: $c_{\boldsymbol{u}}(\texttt{seq}) = \theta_m^{|\boldsymbol{u}|} \sum_{\substack{\boldsymbol{i}=(i_1,\ldots,i_{|\boldsymbol{u}|}) \\ 1 \leq i_1 < \cdots < i_{|\boldsymbol{u}|} \leq |\texttt{seq}|}} \theta_g^{\text{gap}(\boldsymbol{u},\boldsymbol{i})} \mathbb{I}_{\boldsymbol{u}}(\texttt{seq}_{\boldsymbol{i}})$, where $|\boldsymbol{u}|$ is the length of the sub-sequence, $\texttt{seq}_{\boldsymbol{i}} = (\texttt{seq}_{i_1}, \ldots, \texttt{seq}_{i_{|\boldsymbol{u}|}})$, $\text{gap}(\boldsymbol{u}, \boldsymbol{i}) = i_{|\boldsymbol{u}|} - i_1 + 1 - |\boldsymbol{u}|$, and $\mathbb{I}_{\boldsymbol{x}}(\boldsymbol{y})$ is the indicator function assessing if strings $\boldsymbol{x}$ and $\boldsymbol{y}$ match. The match and gap decays $\boldsymbol{\theta} = (\theta_m, \theta_g) \in [0,1]^2$ are learnt from historical data $\mathcal{D}_t = \{\texttt{seq}_i, -\text{QoR}(\texttt{seq}_i)\}_{i=1}^{n_t}$ using Eq. 4 and following projected gradients to ensure feasibility in the $[0,1]^2$ range: $\boldsymbol{\theta}_{\text{update}} = \text{Projection}_{[0,1]^2}(\boldsymbol{\theta}_{\text{current}} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}_{\text{current}}))$, with a step-size η.

Algorithm 1 BOiLS: BO for Logic Synthesis

1: **Input:** Circuit $\mathcal{C}$, maximum number of evaluations $N_{\max}$, maximum number of transformations per sequence K
2: **Initialisation & kernel tuning:**
3: Randomly sample n_0 data to build initial dataset $\mathcal{D}_0 = \{\texttt{seq}_i, -\text{QoR}_{\mathcal{C}}(\texttt{seq}_i)\}_{i=1}^{n_0}$
4: Set the TR radius to $R_{n_0} = K$
5: **Optimisation loop:**
6: **for** $t = 0, \ldots, N_{\max} - 1$ **do**
7: Use $\mathcal{D}_t$ to fit a GP (Section III-B1)
8: Get $\texttt{seq}_{t+1} \in \arg\max_{\texttt{seq} \in \text{TR}(\widehat{\texttt{seq}}_t, \rho_t)} \alpha_{\text{EI}}(\texttt{seq}|\mathcal{D}_t)$
9: Evaluate $\text{QoR}_{\mathcal{C}}(\texttt{seq}_{t+1})$ & augment data
10: Update the TRs radius ρ_{t+1} (Section III-B2)
11: **end for**
12: **Ouptut:** The best sequence of operations $\widehat{\texttt{seq}}_{N_{\max}}$ found

2) Trust-Region Local Search Acquisition Maximisation: BOiLS executes an acquisition maximisation step after fitting a GP with the kernel above, effectively solving $\max_{\texttt{seq} \in \texttt{Alg}^K} \alpha_{\text{EI}}(\texttt{seq}|\mathcal{D}_t)$. To remedy the challenge of solving this combinatorial problem, we equip BOiLS with a local search strategy around an adaptive trust region (TR). At each round t, we use $\widehat{\texttt{seq}}_t$ to denote the best sequence observed so far and define a TR as: $\text{TR}(\widehat{\texttt{seq}}_t, \rho_t) = \{\texttt{seq} \in \texttt{Alg}^K : \text{Hamming}(\widehat{\texttt{seq}}_t, \texttt{seq}) \leq \rho_t\}$, where $\text{Hamming}(\texttt{a}, \texttt{b}) = \sum_i \mathbf{1}_{\texttt{a}_i \neq \texttt{b}_i}$, and ρ_t is an adjustable TR radius scheduled as follows: 1) $\rho_t = \rho_{t-1} + 1$ if we observe 3 improving sequences in a row, 2) $\rho_t = \rho_{t-1} - 1$ if we observe 20 non-improving sequences, or 3) keep ρ_t unchanged otherwise. In case, ρ_t arrives at 0, the TR is empty and the algorithm restarts. With $\text{TR}(\widehat{\texttt{seq}}_t, \rho_t)$ defined, we use the local search strategy from [12] to maximise $\alpha_{\text{EI}}(\texttt{seq}|\mathcal{D}_t)$.

IV. Experimental Results

We assess BOiLS' performance against existing automated and heuristic-based solutions on the 10 circuits of the EPFL arithmetic benchmarks [11].

A. Experimental setup

We run experiments on two machines with Intel Xeon CPU E5-2699 v4@2.20GHz, 64GB RAM, running Ubuntu 18.04.4 LTS and equipped with one NVIDIA Tesla V100 GPU. All algorithms were implemented in Python 3.7 relying on `ABC v1.01`. Area and delay characteristics were measured after FGPA mapping (`if -K 6`) using the `print_stats` command. For each circuit $\mathcal{C}$, we ran BOiLS and alternative methods to solve Eq. (2) with $K = 20$ primitives: `Alg =` [`rewrite`, `rewrite -z`, `refactor`, `refactor -z`, `resub`, `resub -z`, `balance`, `fraig`, `sopb`, `blut`, `dsdb`]. We ran each experiment across 5 random seeds to record statistically significant results for all the large set of optimisers we considered:

- **Deep Reinforcement Learning**: We benchmarked against DRiLLS [8] attempting both PPO and A2C policy update rules, and Graph-RL [9]. We modified the rewards to account for our goal from Eq. (2).
- **Standard Bayesian optimisation (SBO):** To assess the merits of our logic synthesis specific BO method, we also included standard BO from [19].
- **Genetic Algorithm (GA):** We included a standard genetic algorithms from [17] as an additional baseline.
- **Random Search (RS):** Although generally omitted, we add random search as a baseline [16].
- **Greedy Algorithm:** We build a sequence of length K by adding primitives providing the largest immediate QoR gain.
- **EPFL best (count / lvl)**: Those results are the best known solutions achieved for each circuit in [11]. As current heuristics disjointly consider area (*count*) or delay (*lvl*), those aggregated values form a new baseline.

B. Experimental Results

State-of-the-Art QoRs: The table in Fig. 3 reports the best-achieved QoR results across all circuits while restricting the interaction budget $N_{\max} = 200$ across all algorithms. Those values are computed as a relative improvement (in %) compared to `resyn2` using $(\text{QoR}_{\mathcal{C}}(\texttt{resyn2}) - \text{QoR}_{\mathcal{C}}(\widehat{\texttt{seq}}_t))/\text{QoR}_{\mathcal{C}}(\texttt{resyn2})$. BOiLS achieves the best results on average over 8/10 designs and SBO is best in `Log2` circuits and is mostly second to BOiLS. Such results place BO as a

	DRiLLS (PPO)	DRiLLS (A2C)	Graph-RL	GA	RS	Greedy	SBO	BOiLS	EPFL best (lvl)	EPFL best (count)
Adder	22.62	24.59	24.48	24.80	24.27	23.36	25.02	**25.57**	21.36	-55.76
Barrel Shifter	**00.00**	**00.00**	**00.00**	**00.00**	**00.00**	**00.00**	**00.00**	**00.00**	**00.00**	**00.00**
Divisor	40.40	42.66	-	44.82	43.78	40.46	45.49	**47.36**	-59.52	14.04
Hypotenuse	00.81	00.89	-	01.66	01.75	-0.04	01.77	**5.99**	-68.80	01.62
Log2	07.02	07.48	-	07.96	07.77	04.70	**09.01**	08.70	06.25	-33.34
Max	29.28	30.49	31.51	31.97	30.76	28.14	31.04	31.77	**35.61**	-164.0
Multiplier	18.56	19.15	-	20.20	19.25	18.32	20.33	**21.13**	20.67	00.00
Sine	01.64	02.18	01.64	02.70	01.88	00.79	02.64	**03.82**	-23426.71	-26.21
Square-root	12.47	14.07	13.23	13.06	13.70	08.19	13.79	**14.10**	00.00	11.14
Square	36.65	37.77	37.88	38.01	37.78	36.56	38.27	**38.90**	38.88	-21.81
Average	16.94	17.93	-	18.52	18.09	16.05	18.77	**19.74**	-2343	-29.66

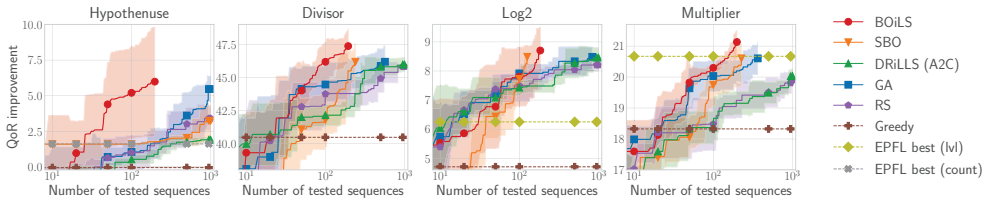

Fig. 2. (Top Row): QoR improvement (in %) for all 10 circuits averaged over 5 random seeds. Note that high computational demands associated with extracting large circuit graphs limit the applicability of graph RL algorithms to small settings. (Bottom Row): Results on the 4 largest circuits showing that BOiLS acquires improved QoR results after about 200 iterations. SBO and GAs present competitive baselines, while DRL rarely outperforms RS strategy.

vital alternative to consider in logic synthesis and show that the modifications from Section III-B further improve performances.

Sample Efficiency: We assess sample efficiency by increasing the budget for all algorithms except for BOiLS. We terminate the loop if methods achieve 97.5% values of BOiLS QoRs or until exhausting a total of $N_{\max} = 1000$ iterations. We report those average results on all 10 circuits in Fig. 1 and on the 5 largest in Fig.3. Clearly: 1) SBO recovers our QoRs but needs 1.5 more trials, 2) GA requires 2.8 times more attempts, and 3) DRL necessitates over 5 times additional sample complexity.

RS as a Valuable Baseline: Our results show that RS is a competitive baseline. We noticed that RS provides similar results to DRL even after 1000 trials. We also ran GA for an $N_{\max}$ of 1000 to assess further improvements. We realised that after 1000 trials, GA attains 4.3% improvement to RS while being $\approx 1\%$ worst than BOiLS. DRL on the other hand, only achieved $\approx 0.12\%$ improvements to RS.

V. Conclusion & Future Work

We proposed BOiLS, the first modern BO solver for LS applications. Our empirical results signify the importance of BO methodologies in LS, demonstrating improved QoR values and reduced sample complexities. Although we chose to optimise QoRs, we note that BOiLS is not tied to a specific black-box and can be utilised with other quantities of interest, e.g., area or delay disjointly by simply modifying Eq. (1).

References

[1] E. Testa, M. Soeken, L. G. Amar *et al.*, " Logic synthesis for established and emerging computing ," Proceedings of the IEEE, 2018.

[2] G. Liu *et al.*, " PIMap: A Flexible Framework for Improving LUT-Based Technology Mapping via Parallelized Iterative Optimization ," ACM Transactions on Reconfigurable Technology and Systems, 2019.

[3] D. Wijerathne *et al.*, " HiMap: Fast and Scalable High-Quality Mapping on CGRA via Hierarchical Abstraction ," in *DATE*, 2021.

[4] S. Ellouz, P. Gamand, C. Kelma, B. Vandewiele, and B. Allard, " Combining internal probing with artificial neural networks for optimal RFIC testing ," IEEE International Test Conference, 2006.

[5] S. Ward, D. Ding, and D. Z. Pan, " PADE: A high-performance placer with automatic datapath extraction and evaluation through high-dimensional data learning ," DAC Design Automation Conference, 2012.

[6] G. Huang *et al.*, " Machine learning for electronic design automation: A survey," ACM Transactions on Design Automation of Electronic Systems.

[7] C. Yu *et al.*, " Developing Synthesis Flows Without Human Knowledge ," Proceedings of the 55th Annual Design Automation Conference, 2018.

[8] H. Abdelrahman, S. Hashemi, M. Shalan, and S. Reda " DRiLLS: Deep reinforcement learning for logic synthesis ," 2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC).

[9] W. Haaswijk *et al.*, " Deep Learning for Logic Optimisation Algorithms ," IEEE International Symposium on Circuits and Systems (ISCAS), 2018.

[10] L. Kaiser, M. Babaeizadeh, P Milos, B. Osinski, R. H. Campbell *et al.*, " Model-based reinforcement learning for atari ," ICLR, 2020.

[11] L. Amaru, P.-E. Gaillardon, and G. De Micheli, " The epfl combinational benchmark suite ," IWLS, no. CONF, 2015.

[12] X. Wan *et al.*, " Think Global and Act Local: Bayesian Optimisation over High-Dimensional Categorical and Mixed Search Spaces ," ICML 2021.

[13] C. Wolf and J. Glaser, " Yosys - A Free Verilog Synthesis Suite " in Proceedings of Austrochip 2013.

[14] C. Yu and W. Zhou, " Decision Making in Synthesis cross Technologies using LSTMs and Transfer Learning ," Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD.

[15] A. Mishchenko *et al.*, " Abc: A system for sequential synthesis and verification ," http://www.eecs.berkeley.edu/alanmi/abc, pp. 1–17, 2007.

[16] J. Blank and K. Deb, " Pymoo: Multi-Objective Optimisation in Python ," IEEE Access, 2020.

[17] D. Pascal, "geneticalgorithm2 (v.6.2.12)", https://github.com/PasaOpasen/geneticalgorithm2, 2021.

[18] B. Shahriari, K. Swersky *et al.*, " Taking the human out of the loop: A review of Bayesian optimization ," Proceedings of the IEEE, 2015.

[19] A. I. Cowen-Rivers, W. Liu *et al.*, "An Empirical Study of Assumptions in Bayesian Optimisation," arXiv preprint arXiv:2012.03826, 2020.

[20] C. E. Rasmussen *et al.*, " Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning) ," The MIT Press, 2005.

[21] J. Močkus, " On Bayesian methods for seeking the extremum ," in Optimization Techniques IFIP Technical Conference, pp. 400–404, Springer, 1975.

[22] H. B. Moss *et al.*, " BOSS: Bayesian Optimization over String Spaces ," in Advances in Neural Information Processing Systems (NeurIPS), 2020.

[23] H. Lodhi, C. Saunders, J. Shawe-Taylor *et al.*, " Text classification using string kernels ," Journal of Machine Learning Research, 2002.