Approximate Logic Synthesis: A Reinforcement Learning-Based Technology Mapping Approach

Ghasem Pasandi, Shahin Nazarian, and Massoud Pedram
Department of Electrical and Computer Engineering
University of Southern California, Los Angeles, CA 90089.
{pasandi, shahin.nazarian, pedram}@usc.edu

Abstract—Approximate Logic Synthesis (ALS) is the process of synthesizing and mapping a given Boolean network to a library of logic cells so that the magnitude/rate of error between outputs of the approximate and initial (exact) Boolean netlists is bounded from above by a predetermined total error threshold. In this paper, we present Q-ALS, a novel framework for ALS with focus on the technology mapping phase. Q-ALS incorporates reinforcement learning and utilizes Boolean difference calculus to estimate the maximum error rate that each node of the given network can tolerate such that the total error rate at non of the outputs of the mapped netlist exceeds a predetermined maximum error rate, and the worst case delay and the total area are minimized. Maximum Hamming Distance (MHD) between exact and approximate truth tables of cuts of each node is used as the error metric. In Q-ALS, a Q-Learning agent is trained with a sufficient number of iterations aiming to select the fittest values of MHD for each node, and in a cut-based technology mapping approach, the best supergates (in terms of delay and area, bounded further by the fittest MHD) are selected towards implementing each node. Experimental results show that having set the required accuracy of 95% at the primary outputs, Q-ALS reduces the total cost in terms of area and delay by up to 70% and 36%, respectively, and also reduces the run-time by 2.21× on average, when compared to the best state-of-the-art academic ALS tools.

I. Introduction

Approximate computing is defined as a computing technique to generate results with possible inaccuracy. This happens by relaxing the exact equivalency requirements between provided specifications and generated results. Approximate computing has attracted tremendous attentions in many different fields that can tolerate inaccuracy, such as video and image processing [1], search engines [2], [3], machine learning [4]–[8], and approximate hardware design [9]–[11] by providing improvement in speed and saving in resources. For example, Esmaeilzadeh et. al [12] accelerated neural networks using approximate computing techniques by introducing a parrot transformation and a neural processing unit.

Approximate computing can be done in different abstraction levels and transformations, one of which is the logic synthesis. Logic synthesis is the process which optimizes and maps a given Boolean network into a netlist consisting of logic gates. The process of exact mapping is present in a variety of synthesis tools. However, in recent times as the demand for energy and area deficiencies has been increasing, there is a need for an alternative approach to meet this demand. This can be achieved by identifying the digital systems which can tolerate errors and arriving at a mapping process which sacrifices accuracy to achieve improvements in area, energy, or delay. This process of synthesis is called Approximate Logic Synthesis (ALS).

There are several papers in the literature for ALS including SALSA [13], SASIMI [14], selection based (Single-Selection and Multi-Selection) algorithms [15] and many more (see Section II), which present new algorithms and/or synthesis tools for generating approximate netlists bounded by a predetermined error rate or error magnitude. These approaches typically suffer from large run-time values. Furthermore, they lack powerful machine learning engines to learn from previous optimization process steps, to better optimize for power, area or delay of the circuit. In this paper, we present Q-ALS, a novel approach for ALS which is based on Reinforcement Learning (RL) algorithms.

Q-ALS embeds a Q-learning agent, which is trained and utilized for determining the maximum error that can be tolerated by a node in a given network such that the total error rate at non of the primary outputs of the network exceeds a predetermined maximum error rate. The Hamming distance between truth tables of exact and approximate implementations of a node is used as the error metric. To estimate the total error rate at primary outputs which is resulted from approximation in a node, a probabilistic approach based on Boolean difference calculus is utilized. Experimental results verify that Q-ALS provides considerable QoR (Quality of Results) improvements in terms of run-time, total area, and the worst case delay of many tested benchmark circuits over other ALS frameworks. Due to the high impact of technology mapping on the final delay and area results, the focus of approximation in this paper is on the technology mapping phase. To the best of our knowledge, this paper is the first to address the problem of approximate logic synthesis using RL algorithms.

The rest of the paper is organized as follows: Section II summarizes some of the previous works on ALS. Section III provides a quick background on Q-learning and cut-based technology mapping, which will be used in the rest of the paper. Section IV presents our Q-ALS framework. Experimental results on many benchmark circuits and conclusions appear in Sections V and VI, respectively.

II. Related Work

Miao *et al.* [16] formulated the ALS problem unconstrained by error rate as a Boolean relations (BR) minimization problem, further refined by a heuristic approach to satisfy the

error frequency constraints. Shin and Gupta [17] developed a method to reduce the literal count by exploiting the error tolerance rate during the circuit design. Miao *et al.* [18] presented a method to reduce the gate count of a given circuit by using the external don't care (EXDC) sets that maximally approach the Boolean relation with compliance to the constrained error magnitude. A novel ALS framework was presented in [19] that performs statistical testing to certify the area optimized circuits with high confidence under the given error constraint. This is done by continuously monitoring the quality of the generated design. In [13], the existing synthesis tools such as SIS [20] and Synopsys Design Compiler [21] are utilized to perform area reduction by using approximate don't cares (ADCs) adhering to the given quality bounds. An ALS tool called SASIMI is presented in [14], which works with multiple error metrics such as error rate and error magnitude. SASIMI aims for area and power reduction by substituting the signal pairs which accept the same values with high probability. Wu and Qian [15] addressed the ALS problem for multi-level circuits by reducing the size of nodes in the given Boolean network with the help of approximating their factored-forms. They presented two algorithms namely Single-Selection and Multi-Selection with similar area savings but better run-time for the latter one.

There have been a few recent publications on the utilization of machine learning algorithms in Electronic Design Automation (EDA). In [22], deep reinforcement learning advances are employed in logic optimization. More specifically, the logic optimization problem is formulated as a deterministic Markov decision process, and a generic algorithm is presented to solve it. In [23], several classical Computer-Aided Design (CAD) algorithms are presented, which can benefit from advances in machine learning. The process of finding the error constraint of each node for large networks leads to diminished performance. The framework which we present in this paper tackles the performance degradation problem by utilizing RL algorithms to find the maximum tolerable error rate at each node. RL, which is shown to resolve such issues [24], will lead to a superior performance in terms of speed, area, and accuracy.

III. Background

A. Q-Learning

Q-learning is an RL algorithm that involves an agent, a set of states, and a set of actions. In Q-learning, at each time t and state s_t, an action a_t is taken, a reward r_t is observed, and the agent enters a new state s_{t+1}. Q-learning algorithm can be modeled as a function that assigns a real number to each pairs of state-actions: $Q : S \times A \rightarrow \mathbb{R}$, and it can be expressed by a Q-matrix. The Q-matrix is updated using the following equation:

$$Q(s_t, a_t) \leftarrow (1 - \alpha) \times Q(s_{t-1}, a_{t-1}) +$$
$$\alpha \times \left(r_t + \gamma \times \max_a Q(s_{t+1}, a) \right) \quad (1)$$

where hyper-parameters α and γ are *learning rate* and *discount factor*, respectively. $\max_a Q(s_{t+1}, a)$ is an estimation for optimal future reward, and $r_t + \gamma \times \max_a Q(s_{t+1}, a)$ is the learned value. Before the learning process starts, the Q-matrix is initialized into random values.

B. Cut-Based Technology Mapping

In state-of-the-art technology mappers [25]–[28], k-$feasible$ cuts [29] are computed for each node in the given Boolean network, and subsequently, truth table of each cut is calculated. A truth table expresses the function of a cut based on its inputs. After computing truth tables, in a topological ordering traversal starting from level 1 nodes, the best cut for each node and the best supergate [28] implementing this cut is computed. A supergate is a small combinational single output network built from original gates in the given library. After visiting all nodes, the best mapping solution (best cover) for the whole network is constructed by traversing the network back and using the computed best cuts and best implementation of those cuts.

IV. Our Proposed Framework for ALS: Q-ALS

Fig. 1 illustrates Q-ALS, our proposed framework for ALS. In our Q-ALS, the cut-based technology mapping approach and the Q-learning algorithm are used to find the best approximate mapping solution for a given Boolean network. Given a maximum error rate at outputs of the given network, a Q-agent is trained using many training networks to learn the maximum error rate that each node can tolerate in order to maximize the delay and area savings. Hamming distance between truth tables of exact and approximate implementations is used as a metric for error rate, and to estimate the propagated error rate to outputs of the network resulted from approximation in a node, a Boolean difference calculus is used, which will be explained later in more details.

For a given training network N, all nodes are visited in a topological ordering traversal and the exact and approximate mapping solutions (matches) to implement the best cut of each node are computed. Approximate matches are computed in two ways: (i) by simply dropping some gates from the exact match (ii) by examining other supergates in the supergate library that can implement function of the cut subject to a given maximum error rate. These approximate supergates are more cost efficient than the exact ones in terms of a desired metric such as delay, area, or power consumption. Similarly to the work in [30], Q-ALS utilizes a prioritized cost function with delay having the highest priority while area is used as a tie breaker.

Fig. 2 shows an example network to demonstrate our method for finding approximate matches in a cut-based technology mapping approach. Fig. 3a shows an exact implementation for the function of cut C shown in Fig. 2. C is a cut among 4-feasible cuts of node n_{13}. The function of this node based on inputs of the shown cut is $n_{13} = e \cdot n_{11} \cdot (n_8 + n_9)$. An approximate implementation with Hamming distance of six between the exact and approximate truth tables is shown in Fig. 3b. By dropping any of the shown inverters from the shown exact implementation, another approximate implementation with the same delay, one unit less area, and Hamming distance of two compared to the exact expression can be found.

A. Action Space

In Q-ALS, states and actions are defined as follows: nodes are considered as states, therefore, there are as many states
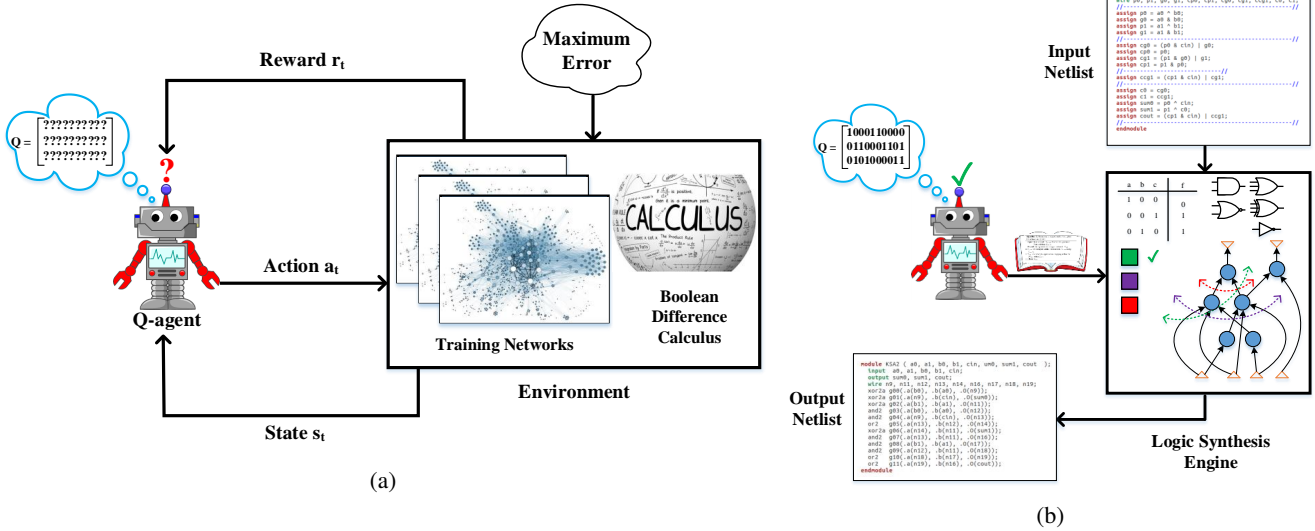
Fig. 1: Our Q-learning based ALS (Q-ALS) framework. (a) The Q-agent of Q-ALS is trained using the provided training networks and given the maximum error rate at primary outputs. (b) At the test time, this Q-agent helps synthesis tool to select the best approximate matches for implementing different nodes.
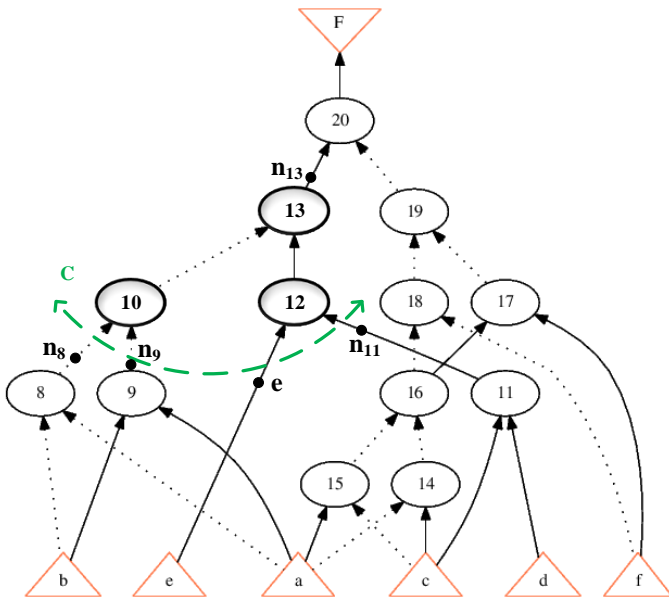

Fig. 2: An example network to demonstrate our method for finding approximate matches in a cut-based technology mapping approach. Function of node n_{13} based on inputs of the shown cut is: $n_{13} = e \cdot n_{11} \cdot (n_8 + n_9)$.
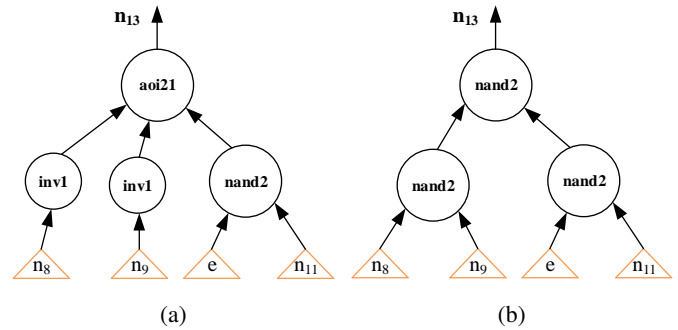

Fig. 3: Two implementations for function of cut C the shown in Fig. 2. (a) an exact implementation with area of 7.00 units and delay of 2.60 units (b) an approximate implementation with area of 6.00 units, delay of 2.00 units, and error rate of 37.5%.

as the total node count in And-Inverter Graph (AIG) [31] representation of the given network. Given the current state as node n, the set of actions that can be taken are selecting a match for n among different exact and approximate solutions for implementing k-feasible cuts of this node. A set of actions which do not generate a valid mapping solution for network N are not desirable. For example, a set of actions is considered invalid, if those actions result in generating a mapping solution for the network N such that the error rate at a primary output of this network exceeds the given maximum error rate.

To estimate the maximum error rate at primary outputs injected by approximate implementation of a node, we used the probabilistic error propagation approach presented in [32]. This approach is based on Boolean difference calculus; it takes as input the Boolean function of a gate, error probabilities at its inputs, and the error probability of the gate itself, and produces the error probability at the output of this gate. We store propagated maximum error rate resulted from approximation in fanin cone of a node into data structure of this node. Next, for each choice of approximate implementation of a node and by using the said error propagation approach, we estimate the error rate at primary outputs of the network.

The Q-agent in Q-ALS learns the maximum error rate a single node can tolerate such that the saving in delay and then area is maximized. If the error rate on a single node is more than this value, it is estimated that it will violate the requirement of maximum error rate at primary outputs. Since the error metric that is employed in Q-ALS is Hamming distance, therefore, the Q-agent in Q-ALS basically learns

the Maximum Hamming Distance (MHD) between exact and approximate truth tables of each node in the network.

B. Reward Function

A reward is assigned to each action based on the saving it offers for delay and area compared to the exact mapping solution. If an approximate match does not improve on the best delay or area of exact implementation, a reward of 0 is assigned to it. On the other hand, if it increases delay and/or area, a negative reward is assigned to it. The goal is to generate a *valid* approximate mapping solution for N which maximizes the total reward.

Fig. 4 illustrates the Q-matrix for a network with six nodes and up to four valid exact and approximate mapping solutions for a single node. This network is for implementing the following function: $F = a \oplus b \oplus c$. One dimension of the corresponding box in this figure represents the nodes of the network, and the other dimension shows the MHD per node. Using the trained Q-agent, the following set of best MHDs are obtained for this network: $\{2, 2, 2, 3, 0, 3\}$. This shows that for example the MHD between truth tables of exact and approximate solutions for node 1 is two. Therefore, a mapping solution with Hamming distance of zero, one, or two can be selected for implementing this node. MHD of zero corresponds to the exact mapping solution.

C. Training

The training process starts with assigning a random integer number between 0 and 32 to MHD of each node. The maximum value of 32 comes from the fact that up to 5-cuts are computed for each node ($32=2^5$). These values are used to start the technology mapping process. If the generated mapping solution based on these MHD values is not valid, then the corresponding entry in the Q-matrix will be reduced. On the other hand, if the assigned MHDs for each node generate a network whose error rate is below the given maximum error rate, then the corresponding positions in the Q-matrix will be increased as long as it does not exceed the maximum

Optimal set of Maximum Hamming
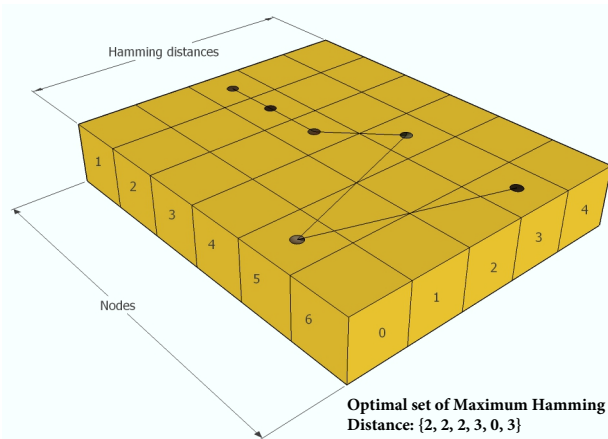Distance: {2, 2, 2, 3, 0, 3}

Fig. 4: A Q-matrix describing optimal set of Maximum Hamming Distance (MHD) for a network with six nodes. This network corresponds to implementing the following function: $F = a \oplus b \oplus c$. Note that $\oplus$ is an XOR operation.

Algorithm 1: Q-ALS model training using Q-learning

Input: Set of training networks: $N_{train} = \{N_1, N_2, ..., N_n\}$,
Maximum error rate: ER_{max}
Output: Coefficients to get Maximum Hamming Distance
(MHD) for a test network.

1 Initialize:
2   Number of Episodes: N_E
3   Learning rate: α
4   Discount factor: γ
5   Q-matrix
6 **for** *each N_i in N_{train}* **do**
7   Get number of nodes in N_i
8   Assign random values for MHD of each node in N_i.
9   **for** *each epoch in N_E* **do**
10   N_{new} = Map N_i using the assigned values for MHD.
11   **if** *mapping is successful* **then**
12   ER_{tmp}=Find the maximum error rate at the outputs of N_{new}.
13   **if** *$ER_{tmp} < ER_{max}$ and $Cost_{N_{new}} < Cost_{N_i}$* **then**
14   Update Q matrix with positive reward.
15   Update $Cost_{N_i}$.
16   **else**
17   Update Q-matrix with negative reward
18   **else**
19   Update Q-matrix with negative reward

20 Coefficients = Perform non-linear regression on Q-matrix
21 **return** *Coefficients*

---

value of 32. In following iterations, only a valid set of values which improves on the previous best delay and area values will update the Q-matrix. This process will continue for N_E times for a particular network and will be repeated for other training networks.

The training process is shown in Algorithm 1. Inputs of this algorithm are training networks and the maximum tolerable error rate at the outputs, ER_{max}. Values for hyper-parameters including number of episodes N_E, the learning rate α, and the discount factor γ are initialized inside this algorithm. Number of episodes is the number of times the training process is executed for a particular network. The Q-matrix generated during the training process will be a 2D matrix where X-axis denotes the nodes in the network and Y-axis denotes the number of options of Hamming distance for the corresponding node. For example, for a network with six nodes, the Q-matrix will have six entries in the X-axis, and a default of 32 entries for the Y-axis.

At the end of the training, we will have a Q-matrix with a certain number of entries in the X-axis, which is equal to the maximum node count among training networks. Now, suppose that the node count in a test network is more than this value, then we cannot use this Q-matrix to test this network. To solve this issue, a non-linear regression is performed to fit a polynomial curve to the data in Q-matrix. Using this fitted curve, MHD values for nodes of any valid test network can be calculated.

Algorithm 2: Generating a mapped netlist for a given test network in Q-ALS

Input: Test network: N_{Test}, Coefficients generated from training: L_{coeff}
Output: An approximate mapped network: N_{opt}

1 L_{HD} <= Compute MHDs for nodes in N_{Test} using L_{coeff}.
2 N_{opt} <= Generate approximate mapping netlist for N_{Test} using L_{HD}.
3 **return** N_{opt}

D. Testing

The conclusion of training process returns coefficients of a polynomial function which will be used to predict the MHD values of a node in the given test network. These predicted MHDs will be used in the logic synthesis engine of Q-ALS to find the best approximate mapping solutions for individual nodes of the test network. The testing pseudo-code is shown in Algorithm 2.

V. EXPERIMENTAL RESULTS

We implemented Q-ALS as an extension to ABC [26]. We first trained our model on the ISCAS 89 [33] and EPFL [34] benchmark circuits and computed entries of the Q-matrix. The maximum error rate at primary outputs used in the experimental results presented in this section is 5%, which is the same as two other state-of-the-art ALS frameworks (i.e., SASIMI and Single/Multi-Selection) compared in this section. The generic standard cell library, *mcnc.genlib*, consisting of 25 gates is used in technology mapping. Test circuits are chosen from different benchmark suites including ISCAS 85 [35], MCNC [36], and ITC 99 [37]. The functionality of circuits widely varies from simple arithmetic circuits of EPFL benchmark suite to complex industry-level circuits of ITC 99 benchmark suite. All experiments were conducted on a virtual machine running Linux with 1GB RAM and a 2.4 GHz laptop as the host machine. Tables I−III contain list experimental results for different circuits. The complexity of circuits in terms of number of gates, exact area, and exact delay is also shown.

In Table I, the 3rd and 4th columns represent exact values of area and delay, respectively. These values are obtained by ABC for benchmark circuits in the first column. The second column represents the number of gates in the corresponding benchmark circuit. Columns five and six represent the area and delay of approximated circuits generated by Q-ALS. Columns seven and eight list area and delay ratios. Area ratio is calculated by dividing the area of an approximate circuit by the area of the corresponding exact circuit. Similarly, the delay ratio is calculated.

Table I shows the area and delay comparison of approximated circuits with their exact counterparts in the MCNC benchmark suite. The MCNC benchmark suite contains different types of circuits such as Finite State Machine (FSM) circuits, multi-level combinational circuits, multi-level sequential circuits, and two-level circuits. Experimenting on these circuits, we get up to 70% area reduction and up to 36%

delay reduction using our Q-ALS. The average area and delay reduction as compared to the baseline values (exact solutions) are 30% and 9%, respectively.

Apart from academic benchmark circuits, we also experimented on industry-level benchmarks from ITC 99 benchmark suite [37]. Table II shows experimental results for this benchmark suite together with a short description for functionality of each circuit. On average for 17 ITC 99 benchmark circuits, Q-ALS provides 23% reduction in area and 10% reduction in delay compared to the exact solution (baseline). This shows that our proposed methodology is quite effective in area and delay reduction for industrial-level circuits as well.

We experimented on benchmark circuits of ISCAS 85 benchmark suite to compare the results of our proposed framework (Q-ALS) with state-of-the-art approximate logic synthesis tool: SASIMI [14] and Single/Multi-Selection approaches [15]. The reason behind choosing ISCAS 85 for this comparison is availability of experimental results in both of these papers for these benchmarks. Table III shows the area comparison between SASIMI, Single/Multi-Selection, and our Q-ALS for ISCAS 85 benchmark circuits. We observed that the average area reduction by SASIMI compared with the exact values is 13.9% and for Single and Multi-Selection are 15.6%, and 15.5%, respectively. The average area reduction by Q-ALS is 31.6% which clearly outperforms other counterparts.

Fig. 5 shows the comparison of run-times of Q-ALS with SASIMI and Single/Multi-Selection. We observed that on average, run-time of Q-ALS as compared to SASIMI, Single-Selection, and Multi-Selection is 15.94×, 8.46×, and 2.21× less, respectively. This is because the inference time of the Q-learning algorithm is very low. The time required to train our model using Q-learning algorithm is around five hours. However, once the model is trained, it can generate the approximated mapping solutions for any given circuit in a small amount of time. In other words, it is trained once, but is used for generating approximate mapping solutions many times. For this reason, the training time is not considered in Fig. 5. Please note that SASIMI and Single/Multi-Selection algorithms are implemented in SIS [20], while Q-ALS is
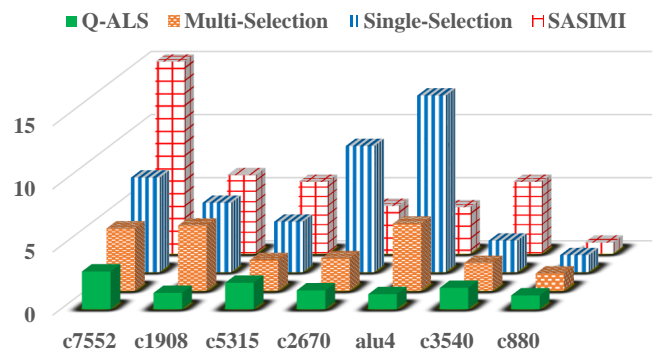

Fig. 5: Run-time (in second) comparison among SASIMI, Single/Multi-Selection, and our Q-ALS for approximate logic synthesis. For better exhibition purposes, data for Single-Selection and SASIMI are scaled down by 5 and 2 times, respectively.

TABLE I: Experimental results for MCNC benchmark suite.

Circuit	Node Count	Exact Area	Exact Delay	Approx. Area	Approx. Delay	Area Ratio	Delay Ratio
rd53	45	107	6.5	54	5.9	0.50	0.91
rd73	115	291	10.5	169	9.7	0.58	0.92
rd84	198	419	10.4	299	9.5	0.71	0.91
9sym	165	418	11.5	360	11.7	0.86	1.02
parity	15	75	7.6	24	4.9	0.32	0.64
my_adder	149	361	36.4	161	26	0.45	0.71
z4ml	38	89	5.2	49	5.2	0.55	1.00
pm1	39	87	4.9	84	5.1	0.97	1.04
c8	122	281	6.9	176	6.4	0.63	0.93
x4	339	760	7.5	760	7.9	1.00	1.05
count	123	273	13.8	171	12	0.63	0.87
pcler8	59	150	7.7	126	7.2	0.84	0.94
sct	68	151	5.5	137	5.7	0.91	1.04
apex7	187	409	12.3	368	10	0.90	0.81

TABLE II: Experimental results for ITC 99 benchmark suite.

Circuit	Function	Node Count	Exact Area	Exact Delay	Approx. Area	Approx. Delay	Area Ratio	Delay Ratio
b01	FSM comparing serial flows	34	86	4.7	56	5.7	0.65	1.21
b02	FSM that recognizes BCS numbers	19	40	5	19	2.9	0.48	0.58
b04	Compute min and max	392	1018	18.6	808	14.8	0.79	0.80
b06	Interrupt handler	34	86	4.1	45	4.3	0.52	1.05
b07	Count points on a straight line	274	671	18.4	559	16.5	0.83	0.90
b08	Find inclusions in sequences of numbers	137	308	12.2	259	10.1	0.84	0.83
b09	Serial to serial converter	118	323	8.3	233	8.6	0.72	1.04
b10	Voting system	142	342	9	276	8.9	0.81	0.99
b11	Scramble string with variable cipher	441	1119	20.1	828	13.7	0.74	0.68
b12	1-player game (guess a sequence)	843	1993	11.8	1642	12.1	0.82	1.03
b13	Interface to meteo sensors	223	517	8.7	372	7.9	0.72	0.91
b14	Viper processor (subset)	4233	10688	44.9	9361	40.4	0.88	0.90
b17	Three copies of 80386 processor (subset)	18446	47667	69.5	41419	60.5	0.87	0.87
b18	Two copies of b14 and two of b17	57978	145896	105.5	136755	105.5	0.94	1.00
b20	A copy and a modified version of b14	8805	21598	53.1	17303	45.6	0.80	0.86
b21	Two copies of b14	9141	22705	52.9	19060	44.8	0.84	0.85
b22	A copy and two modified versions of b14	13561	33260	53.3	26573	46.5	0.80	0.87

implemented in ABC, which is much faster than SIS. To remove this bias, we normalized run-time of SASIMI and Single/Multi Selection approaches by a factor obtained from data published in [28], [38].

VI. CONCLUSION

In this paper, we present Q-ALS, a reinforcement learning based approximate logic synthesis framework. Q-ALS benefits from strong capabilities of Q-learning algorithm to learn the maximum error rate each node in the AIG form of the given network can tolerate in order to achieve a maximum saving in delay and area while bounding to a predetermined error rate at the primary outputs. Thanks to this capability, Q-ALS is able to provide up to 70% area reduction and 36% delay reduction for academic benchmarks, and up to

52% area reduction and 30% delay reduction for industrial-level benchmarks. Furthermore, Q-ALS reduces run-time by an average of $15.94\times$ and $2.21\times$ over SASIMI and Multi-Selection, two academic state-of-the-art ALS tools.

REFERENCES

[1] J. Huang, B. Wang, W. Wang, and P. Sen, "A surface approximation method for image and video correspondences," *IEEE Transactions on Image Processing*, vol. 24, no. 12, pp. 5100–5113, 2015.

[2] G. Ranjan, A. Tongaonkar, and R. Torres, "Approximate matching of persistent lexicon using search-engines for classifying mobile app traffic," in *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. IEEE, 2016, pp. 1–9.

[3] K.-H. Yang, C.-C. Pan, and T.-L. Lee, "Approximate search engine optimization for directory service," in *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*. IEEE, 2003, pp. 8–pp.

[4] A. J. Smola and B. Schölkopf, "Sparse greedy matrix approximation for machine learning," 2000.

TABLE III: Experimental results for ISCAS 85 benchmark suite.

Circuit	Exact Area	SASIMI [14]		Single-Selection [15]		Multi-Selection [15]		Q-ALS	
		Approx. Area	Area ratio	Approx. Area	Area ratio	Approx. Area	Area ratio	Approx. Area	Area ratio
c880	646	579	0.896	577	0.893	577	0.893	558	0.864
c1908	846	516	0.610	503	0.595	506	0.598	520	0.615
c2670	1298	940	0.724	859	0.662	874	0.673	866	0.667
c3540	1916	1868	0.975	1851	0.966	1849	0.965	1812	0.945
c5315	3060	3002	0.981	2993	0.978	3002	0.981	1901	0.621
c7552	3952	3746	0.948	3715	0.940	3719	0.941	2451	0.620
alu4	2740	2444	0.892	2406	0.878	2381	0.869	1260	0.460

[5] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*, 2016, pp. 1050–1059.

[6] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," *arXiv preprint arXiv:1401.4082*, 2014.

[7] J. Li, Z. Yuan, Z. Li, A. Ren, C. Ding, J. Draper, S. Nazarian, Q. Qiu, B. Yuan, and Y. Wang, "Normalization and dropout for stochastic computing-based deep convolutional neural networks," *Integration, the VLSI Journal*, 2017.

[8] M. Nazemi, G. Pasandi, and M. Pedram, "Energy-efficient, low-latency realization of neural networks through boolean logic minimization," in *24th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2019, pp. 1–6.

[9] S. Dutt, S. Nandi, and G. Trivedi, "A comparative survey of approximate adders," in *Radioelektronika (RADIOELEKTRONIKA), 2016 26th International Conference*. IEEE, 2016, pp. 61–65.

[10] M. Masadeh, O. Hasan, and S. Tahar, "Comparative study of approximate multipliers," *arXiv preprint arXiv:1803.06587*, 2018.

[11] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasicek, and K. Roy, "Design of power-efficient approximate multipliers for approximate artificial neural networks," in *Proceedings of the 35th International Conference on Computer-Aided Design*. ACM, 2016, p. 81.

[12] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE Computer Society, 2012, pp. 449–460.

[13] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "Salsa: systematic logic synthesis of approximate circuits," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 796–801.

[14] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2013, pp. 1367–1372.

[15] Y. Wu and W. Qian, "An efficient method for multi-level approximate logic synthesis under error rate constraint," in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016, p. 128.

[16] J. Miao, A. Gerstlauer, and M. Orshansky, "Approximate logic synthesis under general error magnitude and frequency constraints," in *Proceedings of the International Conference on Computer-Aided Design*. IEEE Press, 2013, pp. 779–786.

[17] D. Shin and S. K. Gupta, "Approximate logic synthesis for error tolerant applications," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2010, pp. 957–960.

[18] J. Miao, A. Gerstlauer, and M. Orshansky, "Multi-level approximate logic synthesis under general error constraints," in *Computer-Aided Design (ICCAD), 2014 IEEE/ACM International Conference on*. IEEE, 2014, pp. 504–510.

[19] G. Liu and Z. Zhang, "Statistically certified approximate logic synthesis," in *Computer-Aided Design (ICCAD), 2017 IEEE/ACM International Conference on*. IEEE, 2017, pp. 344–351.

[20] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," 1992.

[21] Synopsys, "Design compiler user guide," *Inc., see http://www. synopsys. com*, 2001.

[22] W. Haaswijk, E. Collins, B. Seguin, M. Soeken, F. Kaplan, S. Süsstrunk, and G. De Micheli, "Deep learning for logic optimization algorithms," in *Circuits and Systems (ISCAS), 2018 IEEE International Symposium on*. IEEE, 2018, pp. 1–4.

[23] P. A. Beerel and M. Pedram, "Opportunities for machine learning in electronic design automation," in *Circuits and Systems (ISCAS), 2018 IEEE International Symposium on*. IEEE, 2018, pp. 1–5.

[24] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana, "Self-optimizing memory controllers: A reinforcement learning approach," in *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3. IEEE Computer Society, 2008, pp. 39–50.

[25] G. Pasandi and M. Pedram, "PBMap: A path balancing technology mapping algorithm for single flux quantum logic circuits," *IEEE Transactions on Applied Superconductivity*, vol. 29, no. 4, pp. 1–14, 2019.

[26] U. Berkeley, "ABC: A system for sequential synthesis and verification," *Berkeley Logic Synthesis and Verification Group*, 2011.

[27] G. Pasandi, A. Shafaei, and M. Pedram, "SFQmap: A technology mapping tool for single flux quantum logic circuits," in *Circuits and Systems (ISCAS), 2018 IEEE International Symposium on*. IEEE, 2018, pp. 1–5.

[28] A. Mishchenko, S. Chatterjee, R. Brayton, X. Wang, and T. Kam, "Technology mapping with boolean matching, supergates and choices," 2005.

[29] J. Cong and Y. Ding, "FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based fpga designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 1, pp. 1–12, 1994.

[30] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts," in *Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design*. IEEE Press, 2007, pp. 354–361.

[31] A. Mishchenko, R. Jiang, S. Chatterjee, and R. Brayton, "Fraigs: Functionally reduced and-inv graphs," in *International Conference on Computer Aided Design*, 2004.

[32] N. Mohyuddin, E. Pakbaznia, and M. Pedram, "Probabilistic error propagation in a logic circuit using the boolean difference calculus," in *Advanced Techniques in Logic Synthesis, Optimizations and Applications*. Springer, 2011, pp. 359–381.

[33] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Circuits and Systems, 1989., IEEE International Symposium on*, May 1989, pp. 1929–1934 vol.3.

[34] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, no. EPFL-CONF-207551, 2015.

[35] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran," in *Proceedings of IEEE Int'l Symposium Circuits and Systems (ISCAS 85)*. IEEE Press, Piscataway, N.J., 1985, pp. 677–692.

[36] S. Yang, *Logic synthesis and optimization benchmarks user guide: version 3.0.* Microelectronics Center of North Carolina (MCNC), 1991.

[37] F. Corno, M. S. Reorda, and G. Squillero, "Rt-level itc'99 benchmarks and first atpg results," *IEEE Design & Test of computers*, vol. 17, no. 3, pp. 44–53, 2000.

[38] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG rewriting a fresh look at combinational logic synthesis," in *Proceedings of the 43rd annual Design Automation Conference*. ACM, 2006, pp. 532–535.