Rank-based Multi-objective Approximate Logic Synthesis via Monte Carlo Tree Search

Yuyang Ye[1], Xiangfei Hu[2], Yuchen Liu[2], Peng Xu[1], Yu Gong[3], Tinghuan Chen[4], Hao Yan[2], Bei Yu[1], Longxing Shi[2]
[1]CUHK [2]Southeast University [3]Nanjing University of Aeronautics and Astronautics [4]CUHK-Shenzhen

***Abstract*—Approximate Logic Synthesis (ALS) is an automated technique designed for error-tolerant applications, optimizing delay, area, and power under specified error constraints. However, existing methods typically focus on either delay reduction or area minimization, often leading to local optima in multi-objective optimization. This paper proposes a rank-based multi-objective ALS framework using Monte Carlo Tree Search (MCTS). It develops non-dominated circuit ranking, to guide MCTS in exploring local approximate changes (LACs) across the entire circuit and generate approximate circuit sets with great optimization potential. Additionally, a Rank-Transformer model is introduced to predict path-domain ranks, enhancing the application of high-quality LACs within circuit paths. Experimental results show that our framework achieves faster and more efficient optimization in delay and area simultaneously compared to state-of-the-art methods.**

I. Introduction

As transistor technology advances into the nanoscale, optimizing circuit timing and area has become critical. With the growing demand for error-tolerant applications such as image processing and machine learning, approximate computing has emerged as an innovative circuit design paradigm [1]. By carefully managing introduced errors, it enables significant reductions in circuit delay and area with minimal impact on functionality.

To automatically generate approximate circuits that meet user-defined error constraints, Approximate Logic Synthesis (ALS) techniques have been developed. These methods optimize circuits by applying Local Approximate Changes (LACs) on circuits, which modify local circuit structures. Based on their optimization targets, ALS methods are categorized into two types: (1) **Delay-driven methods:** These methods iteratively apply LACs to critical paths, reducing the propagation delay of specific gates and improving overall circuit timing. For instance, HEDALS [1] and TCAD24 [2] use critical error graphs and reinforcement learning to accelerate this process with minimal error. DCGWO [3] reduces both critical path depth and area while enhancing gate drive strength, further decreasing delay. (2) **Area-driven methods:** These methods, including [4]–[9], treat all circuit gates in circuits as LAC candidates, prioritizing them that remove the most gates. Through multiple iterations, they identify LACs that maximize area reduction.

However, existing chips commonly pursue higher frequency and lower area usage. This requires designers to optimize both delay and area simultaneously. The LACs used in delay-driven methods are overly localized, as they are confined to critical paths and adjacent paths of them. Meanwhile, the LACs used in area-driven methods are too global, as they are scattered across the entire circuit. As a result, both methods fail to adequately balance the global and local search for LACs, leading to insufficient multi-objective optimization.

When dealing with a multi-objective ALS problem with an extremely large solution space, Evolutionary Algorithms [10] and Bayesian Optimization [11] struggle to find the true Pareto frontier within an acceptable cost. In this case, LAMOO [12] recommends using Monte Carlo Tree Search (MCTS) to assist in learning space partitioning. By sampling within the finely partitioned solution space, LAMOO can quickly find optimal solutions. The method relies on accurate performance modeling for effective partitioning and searching. However, it is challenging to obtain accurate models for large circuits, which is considered in our work. The inaccuracy easily causes conventional MCTS to sample in incorrect solution spaces, leading to local optima.

To address the aforementioned challenges, we propose a rank-based multi-objective Approximate Logic Synthesis (ALS) framework leveraging Monte Carlo Tree Search (MCTS). This framework integrates domain-specific knowledge in ALS to customize and implement MCTS, achieving an effective balance between global and local search for Logic Approximation Candidates (LACs). By incorporating circuit-domain and path-domain ranking mechanisms, the framework guides MCTS exploration, enabling efficient simultaneous optimization of delay and area without relying on precise modeling. Our main contributions are summarized as follows:

- We propose a multi-objective ALS framework based on tailored MCTS. It can optimize delay and area under various error constraints.
- We design a non-dominated circuit ranking method to split LAC solution space and to guide MCTS's global exploration. This method identifies LAC-induced approximate circuits with high optimization potential of delay and area for subsequent sampling.
- We develop a Rank-Transformer to predict path-domain LAC ranks, steering MCTS toward high-quality local LACs on critical paths, facilitating more effective delay and area optimization.
- On open-source circuits using the TSMC 28nm library, our framework achieves superior delay reduction and area savings under different error constraints, with some speedup compared to state-of-the-art methods.

II. Preliminaries

A. Error Metrics

Our framework supports two commonly used error metrics: normalized mean error distance (NMED) and error rate (ER). For a circuit with respectively I and O numbers of PIs and POs, assume that the probability of input vector I_i occurring is P_i ($1 \leq i \leq 2^I$). In this case, NMED and ER can be defined as follows:

Mean error distance is the mean difference between approximate circuit output value V_i^{app} and accurate circuit output value V_i^{acc}. NMED is the mean error distance normalized by the maximum output value, defined in Equation (1).

$$\text{NMED} = \sum_{i=1}^{2^I} \frac{P_i \times |V_i^{\text{app}} - V_i^{\text{acc}}|}{2^O - 1}. \tag{1}$$

ER is the total probability of incorrect outputs generated by the input patterns of an approximate circuit. It is calculated by Equation (2), where O_i^{app} and O_i^{acc} are output vectors of the approximate circuit and accurate circuit for input vector I_i.

$$\text{ER} = \sum_{i=1}^{2^I} \left[P_i \times (O_i^{\text{app}} \neq O_i^{\text{acc}})\right]. \tag{2}$$

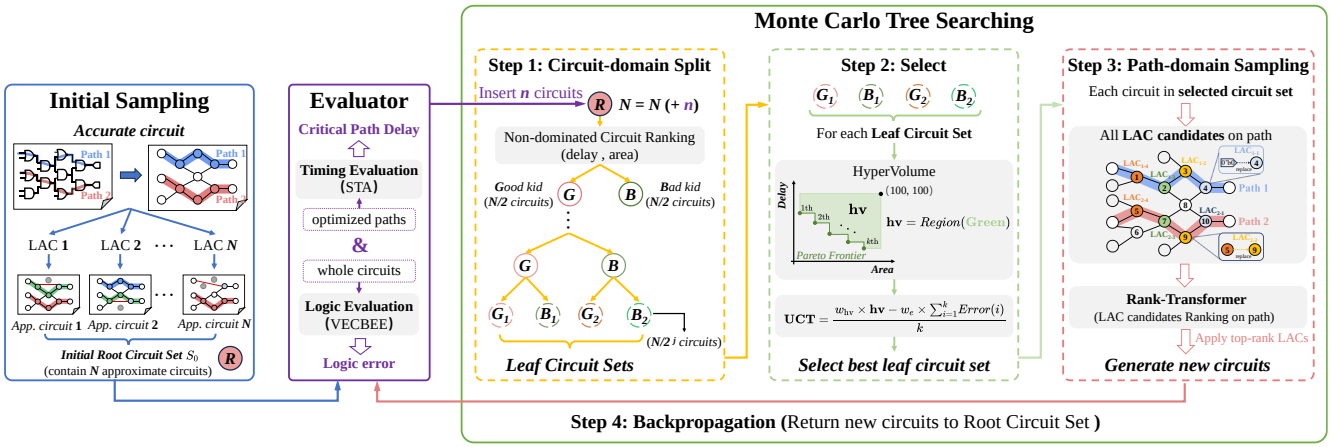


Fig. 1 The overall flow of Rank-based Multi-objective Approximate Logic Synthesis framework via Monte Carlo Tree Search.

B. Timing and Logic Error Evaluator

Timing Evaluation. We use the critical path delay, which determines the minimum specified clock period, to measure the timing performances of circuits. The critical path delay of the approximate circuit is obtained through Static Timing Analysis using Synopsys PrimeTime [13].

Logic Error Evaluation. Instead of the traditional full-input pattern traversal method, we use VECBEE [8] based on Monte Carlo simulation to evaluate the logic error of the approximate circuit. This method can accelerate the capture of signal changes caused by LACs, achieving fast error evaluation with nearly no deviation.

C. Problem Formulation

Each LAC used in our work is gate-based, defined as follows:

Definition 1 (Gate-based LAC)**.** *The LAC that substitutes a gate in the circuit with another gate or a constant logic value '0' or '1'. The replaced gate is called the target gate, and the gate (or value) replacing it is called the switch gate.*

Based on **Definition 1**, the rank-based ALS problem can be formulated as follows:

Problem 1 (Rank-based ALS)**.** *Given a post-synthesis accurate circuit with timing and logic information, select and apply proper LACs based on the rank of the explored gate-based LACs to deeply optimize its delay and area under the error constraint.*

III. Proposed Framework

The overall flow of our framework is shown in Fig. 1. The framework uses MCTS to deeply explore the optimization potential of delay and area. Each node in the search tree represents a circuit set, containing multiple approximate circuits. The MCTS receives initial circuits and iteratively performs four steps: *Split*, *Selection*, *Sampling* and *Backpropagation*. In each iteration, *Split* dynamically modifies the tree structure from the root circuit set. *Selection* identifies a suitable leaf circuit set, while *Sampling* uses gate-based LACs to perform further optimization on the circuits within the selected leaf circuit set. The new circuits are *backpropagated* to the root circuit set after timing and logic evaluation. This iterative process ensures efficient exploration and optimization across the approximate circuit space.

A. Initial Sampling

The accurate circuit is represented by a directed acyclic graph (DAG). In this DAG, nodes are gates and edges are connection relationships between gates. To generate circuits for MCTS optimization, we applied N different LAC samplings separately to the accurate circuit.

Within each LAC sampling, the target gate is selected from the critical paths. Then, the switch gate is filtered from gates with shorter arrival time in the path. To limit the introduced error, the gate having the highest **output similarity** with the target gate is preferred. Assume that $O_{I_i}^{\text{tar}}$ and $O_{I_i}^{\text{swi}}$ are respectively output signal of the target gate and the switch gate candidate under input vector I_i, the output similarity SIM can be calculated in Equation (3):

$$\text{SIM} = \sum_{i=1}^{N_I} \left[\frac{(O_{I_i}^{\text{tar}} == O_{I_i}^{\text{swi}})}{N_I} \right], \tag{3}$$

where N_I is the number of input vectors. After evaluation, the generated circuits without error violations are inserted into the root circuit set for MCTS.

B. Circuit-domain Rank-based Splitting

The split step is designed to achieve the partitioning of the current root circuit set $S_t : \{c_1^t, c_2^t, \ldots, c_n^t\}$ and reconstruct a performance-driven search tree. To cover the solution space of **Problem 1**, an exponential number of circuits are required. After circuit-domain splitting, the vast solution space (i.e., the root circuit set S_t) is partitioned and performance-driven promising circuit sets (i.e., leaf circuit sets) are generated. In leaf circuit sets, the timing and area performance of circuits are better and the optimization potential is greater. As capturing an accurate performance model and optimization gradient is challenging, we propose a non-dominated circuit ranking to sort the circuits in S_t. This ranking process consists of two parts: Pareto level ranking, and crowding distance ranking within one specific Pareto level.

Pareto Level Ranking. In Pareto level ranking, we first establish the Pareto dominance between circuits in the circuit set. If c_i^t is not worse than c_j^t in both delay and area, and strictly better in at least one, then c_i^t dominates c_j^t. Circuits not dominated by any others are assigned to the 0-th Pareto level (i.e., their Pareto-level PL $= 0$). Subsequently, circuits dominated only by circuits in previous Pareto levels are assigned to the next level. This process repeats until all circuits are assigned. Taking Fig. 2 as an example, approximate circuit A is not dominated by any other circuit, thus its Pareto-level $\text{PL}_A = 0$. Since B and C are dominated only by circuits in the 0-th Pareto level, they are assigned to the 1-th Pareto level. Additionally, D and E are dominated only by circuits in the 0-th and 1-th Pareto level, thus they are assigned to the 2-th Pareto level. Circuits with lower

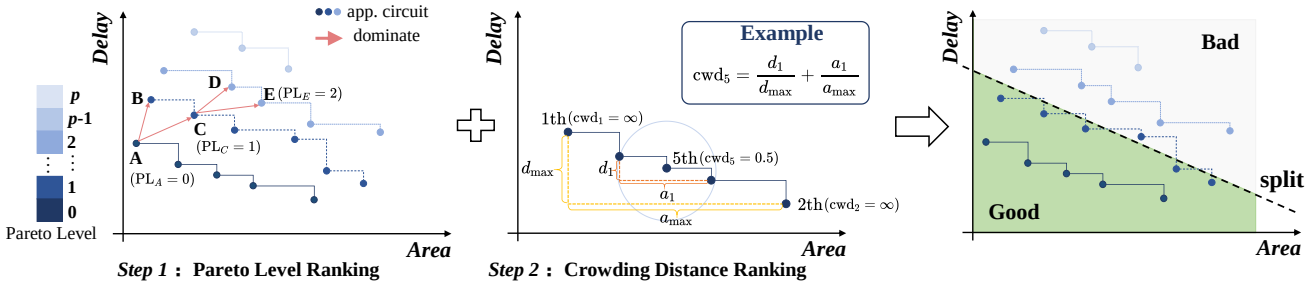


Fig. 2 The illustration of non-dominated circuit ranking. Step 1: Coarse ranking based on Pareto Level. Step 2: Precise ranking based on crowding distance within each Pareto Level.

Pareto-level PL indicate better timing and area performance.

Crowding Distance Ranking. Within each Pareto level, we perform further ranking based on the crowding distance. For circuits with higher crowding distances, conducting further sampling on them can lead to more efficient exploration. It is because their gradients are less likely to overlap in the solution space. Therefore, we sort the circuits within the same Pareto level in descending order of crowding distance. As shown in the Fig. 2, in the i-th Pareto level, the largest delay difference and area difference between circuits are $d_{\max}$ and $a_{\max}$, respectively. For circuit c_j^i in this Pareto level, the adjacent circuits are c_{j-1}^i and c_{j+1}^i. Assume that the delay difference and area difference between c_{j-1}^i and c_{j+1}^i are d_j and a_j, respectively. In this case, the crowding distance of c_j^i can be calculated as follow:

$$\mathrm{cwd}(c_j^t) = \frac{d_j}{d_{\max}} + \frac{a_j}{a_{\max}}. \tag{4}$$

Since the circuits at both ends of the Pareto level have no adjacent circuits, their crowding distances are set to infinity ($+\infty$). In the case of identical crowding distances, we consider circuits with a smaller delay to be more worthy of further sampling. The overall rank is referred to as the circuit-domain ranking and used to replace gradient prediction for space partitioning.

Splitting based on Circuit-domain Ranking. The MCTS partitions S_t and reconstructs the search tree based on the circuit-domain ranking. At the beginning of the iteration, the search tree will be reset to the root circuit set S_t. The non-dominated circuit ranking is performed on S_t to get the circuit rank. Circuits in the top 50% rank with better performance and greater optimization potential are labeled as positive while the others are negative. Subsequently, two leaf circuit sets are expanded from S_t, including a good kid set that receives the positive circuits and a bad kid set that receives the negative circuits. The generated good kid set also undergoes the aforementioned split and expansion. This process continues until the number of circuits inside all newly generated leaf circuit sets is less than a user-specified value. In the final rounds of the process, the newly generated bad kid set also experiences split and expansion to ensure the diversity of the partitioned spaces.

C. Selection

The Selection step is an extension of the circuit-domain rank. It incorporates accumulated logic errors from the iterative optimization process to guide a more precise search. Specifically, it filters the best circuit set based on the upper confidence bound for trees (UCT) from the leaf circuit sets of the search tree generated after splitting. Since ALS requires sufficient optimization of the objectives with minimal error, UCT is computed based on the HyperVolume $\mathbf{hv}$ and the average error of k Pareto front circuits within the circuit set.

HyperVolume $\mathbf{hv}$ can provide an effective evaluation of the convergence, uniformity, and spread of samples when the true Pareto frontier cannot be determined [12]. Therefore, it is highly suitable for our optimization problem. Higher $\mathbf{hv}$ characterizes better optimization quality of the Pareto front circuits within the circuit set. Assuming the x-coordinate of the objective space represents the ratio of i-th approximate circuit area to accurate circuit area $Ratio_a^{(i)}$, and the y-coordinate represents the corresponding delay ratio $Ratio_d^{(i)}$. In this case, the reference point is (1, 1). $\mathbf{hv}$ is the area enclosed by k Pareto front circuits and the reference point, as defined in Equation (5). Note that $Ratio_a^{(0)} = 1$. For one leaf circuit set, it is computed as:

$$\mathbf{hv} = \sum_{i=1}^{k} \left[(1 - Ratio_d^{(i)}) \times (Ratio_a^{(i-1)} - Ratio_a^{(i)}) \right]. \tag{5}$$

The average error is introduced to ensure the approximated circuits are under the error constraint. Based on HyperVolume $\mathbf{hv}$ and the average error, UCT of one leaf circuit set is computed as:

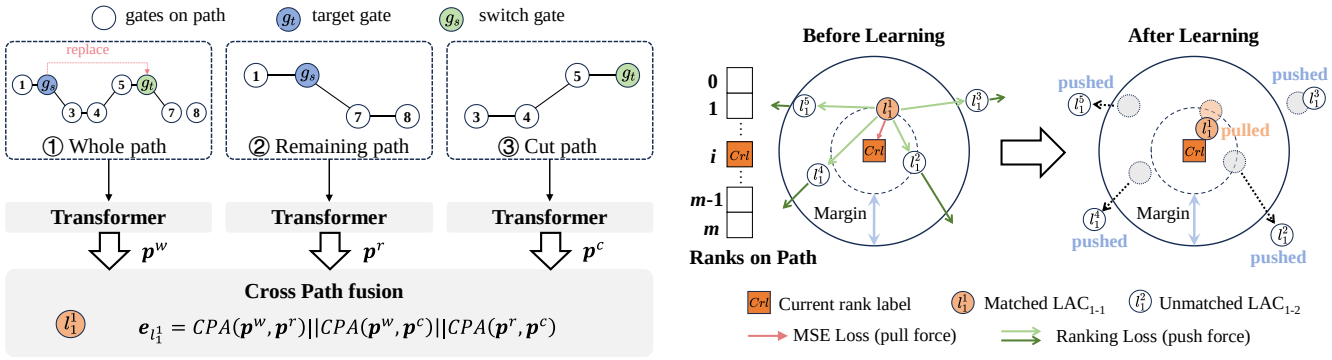
$$\mathbf{UCT} = \frac{w_{hv} \times \mathbf{hv} - w_e \times \sum_{i=1}^{k} Error(i)}{k}, \tag{6}$$

where w_{hv} and w_e are weights of HyperVolume and average error, respectively. The leaf circuit set with the highest UCT is selected for further sampling optimization.

D. Path-domain Rank-based Sampling

The sampling is performed on all circuits inside the selected leaf circuit set. In our MCTS framework, we maintain and apply a set of timing-area-reducing LAC $\mathcal{L}_M : \{l_M^1, l_M^2, \ldots, l_M^n\}$ for each circuit to generate new approximate circuits. To ensure that applying $\mathcal{L}_M$ can bring both stable delay and area reduction, each LAC in $\mathcal{L}_M$ is extracted from critical paths of the circuit. In a given critical path with $K+1$ gates, we consider all gates except those directly connected to PI as target gate candidates. For each candidate, we select a gate in its preceding path with the highest similarity SIM as the switch gate, thereby forming a gate-based LAC candidate l_C. This approach yields a critical path LAC set $\mathcal{L}_C : \{l_c^1, l_c^2, \ldots, l_c^K\}$. To select high-quality l_c from $\mathcal{L}_C$ to insert into $\mathcal{L}_M$. Thus, the LAC candidates in $\mathcal{L}_C$ need to be ranked based on their induced delay and area improvements. As ranking through simulation incurs high time costs, we design and implement a Rank-Transformer to enable fast rank predictions on paths. Its detailed information includes feature engineering, embedding & fusion and rank prediction.

Feature Engineering. As shown in Fig. 3a, for each l_c, we create three paths based on its target gate g_t and switch gate g_s, including: (1) the whole path, which retains the complete information of the path before approximation; (2) the cut path, which contains information of removed gates; (3) the remaining path, which includes information of



(a) Path embedding and fusion.

(b) Training of rank prediction.

Fig. 3 The illustration of the Rank-Transformer. In (b), matched LAC refers to LAC whose rank within a path matches the true rank label, while unmatched LAC refers to LAC with mismatched rank. The pull is driven by MSE Loss, and the push is induced by Ranking Loss.

the path after approximation. These three paths can effectively capture the impact of approximation on the entire critical path, enabling the rank-transformer to comprehensively learn path-domain information.

Additionally, three parameters influencing the optimization brought by l_c are also provided and regarded as the feature of l_c, including delay reward Reward_D, area reward Reward_A and similarity SIM. SIM is already defined in Section III-A and is computed based on Equation (3). Reward_D refers to the difference between the maximum delays of g_t and the g_s. Reward_A refers to the total area of gates inside the difference set between the maximum fanout free cones (MFFCs) of g_t and g_s. These two rewards are defined as follows:

$$\text{Reward}: \begin{cases} \text{Reward}_D = \text{Delay}(g_t) - \text{Delay}(g_s) \\ \text{Reward}_A = \text{Area}(MFFC_{g_t} - MFFC_{g_s}) \end{cases}, \quad (7)$$

.

Embedding & Fusion. The embedding result of each LAC candidate $\boldsymbol{e}_{l_c}$ is generated based on fusion embedding results of the whole path, the cut path and the remaining path. Transformer helps us to embed each path independently and generates the whole path embedding result $\boldsymbol{p}_{l_c}^w$, the cut path embedding result $\boldsymbol{p}_{l_c}^c$ and the remaining path embedding result $\boldsymbol{p}_{l_c}^r$. As shown in Fig. 3a, a cross-path attention mechanism (CPA) proposed in [14] is then employed to achieve path fusion. In our work, the self-attention in Transformer helps to capture internal-path relationships and the cross-path attention mechanism facilitates information exchange among different paths. Finally, the final embedding result of LAC candidate $\boldsymbol{e}_{l_c}$ can be computed by contacting the three path embedding results as:

$$\boldsymbol{e}_{l_c} = \text{CPA}(\boldsymbol{p}_{l_c}^w, \boldsymbol{p}_{l_c}^c) \| \text{CPA}(\boldsymbol{p}_{l_c}^w, \boldsymbol{p}_{l_c}^r) \| \text{CPA}(\boldsymbol{p}_{l_c}^c, \boldsymbol{p}_{l_c}^r) \quad (8)$$

Rank Prediction. Based on the final embedding result of LAC candidate $\boldsymbol{e}_{l_c}$, we can predict the rank of it using the Softplus function as:

$$r_c = \text{Softplus}(\boldsymbol{e}_{l_c}). \quad (9)$$

Inspired by Ranknet [15], we design a loss function $\mathcal{L}$ in this work to improve the accuracy of predicted LAC ranks on the path. $\mathcal{L}$ is defined as follows:

$$\mathcal{L} = \mathcal{L}_{MSE} + \mathcal{L}_{rank}, \quad (10)$$

where $\mathcal{L}_{MSE}$ and $\mathcal{L}_{rank}$ are MSE loss and ranking loss respectively. For a given LAC candidate l_c, assume the true rank label of it is $\hat{r}_c$. In this case, the MSE loss is the mean squared difference between the predicted ranks and the true rank labels:

$$\mathcal{L}_{MSE} = \frac{1}{K} \sum_{i=1}^{K} (r_c - \hat{r}_c)^2. \quad (11)$$

As shown in Fig. 3b, for a given true rank label, the MSE loss $\mathcal{L}_{MSE}$ pulls the predicted rank of matched LAC (i.e., LAC whose true rank match the current true rank label) closer to it. Since relying solely on the pull force of the MSE Loss for ranking can lead to the homogenization of predicted ranks, the ranking loss $\mathcal{L}_{rank}$ is introduced for correction. $\mathcal{L}_{rank}$ is a customized pair-wise loss built upon the comparison between the predicted ranks of l_c and l_j:

$$\mathcal{L}_{rank} = \frac{2}{K^2 - 1} \sum_{i=1}^{K} \max(0, R_c^i), \quad (12)$$

$$R_c^i = \text{Margin} - \text{sign}[(r_c - r_i) \cdot (\hat{r}_c - \hat{r}_i)], \quad (13)$$

where Margin is a hyper-parameter. Fig. 3b illustrates that the ranking loss $\mathcal{L}_{rank}$ generates a push force between the predicted ranks of matched LAC and unmatched LACs (i.e., LACs whose true ranks do not match the current true label rank), pushing the latter at least by a Margin difference away from the former. Therefore, the predicted ranks of LAC candidates in $\mathcal{L}_C$ can maintain certain differences, effectively preventing ranking homogenization.

By applying the above rank prediction mechanism to the path, the rank of LAC candidates in $\mathcal{L}_C$ (i.e., the **path-domain rank**) are effectively obtained. LAC candidates with top ranks are selected into $\mathcal{L}_M$ and applied. In the later iterations of MCTS, the number of critical paths in the circuit may become extremely large. When several rounds of search fail to effectively reduce the delay, a set of area-reducing LAC $\mathcal{L}_A : \{l_A^1, l_A^2, \ldots, l_A^n\}$ for each circuit is maintained to replace $\mathcal{L}_M$. Then, LACs in $\mathcal{L}_A$ are chosen by a greedy algorithm.

E. Backpropagation

As the length of LACs sequences applied to the accurate circuit increases with iterations, the performances of the newly generated circuits, including their delay reductions, area reductions and introduced logic errors, can effectively represent the impact brought by current optimization process. Therefore, these newly generated circuits are backpropagated to the root circuit set S_t after evaluation, expanding S_t to S_{t+1}. Note that circuits with error violations are discarded.

TABLE I Statistics of the benchmarks used in our experiment. The units of delay and area are respectively ps and μm^2.

Random / Control				Arithmetic			
Circuit	#Gate	Delay	Area	Circuit	#Gate	Delay	Area
c880	322	185.34	177.67	c6288	1641	847.79	687.08
c1908	366	235.14	223.34	adder	1639	1394.7	495.78
c2670	922	218.40	288.71	barshift	2933	262.52	1806.6
c3540	667	293.09	459.42	max	2940	2799.8	954.03
c5315	2595	122.25	1129.6	mult	26429	4117.5	31635.6
c7552	1576	282.13	939.33	sine	11560	3234.4	7173.9
cavlc	573	186.35	450.31	sqrt	13542	67929.3	6262.1
priority	2336	1126.8	1423.3	square	14696	8211.1	7752.8

IV. Experimental Results

A. Experiment Setup

Our framework is implemented in C++ and the Rank-Transformer is implemented in Python using the PyTorch library. Our work is tested on the Linux machine with 32 cores and 4 NVIDIA Tesla V100 GPUs. The LAC candidates used for the Rank-Transformer training are derived from TSMC 28nm standard arithmetic circuits, including different bit-width adders, multipliers, and divisors.

Our framework is applied to circuits selected from ISCAS'85 [16] and EPFL [17] to demonstrate its effectiveness and generalization capability. These circuits are pre-synthesized to gate-netlists with TSMC 28nm technology using Synopsys Design Compiler [18]. Their statistics are listed in TABLE I. Among them, the random/control circuits are optimized under ER constraints, while the arithmetic circuits are optimized under NMED constraints. For each generated approximate circuit, we use Synopsys PrimeTime [13] to extract its critical paths and report the delay. Meanwhile, the circuit area is obtained using ABC [19]. In terms of logic simulation, we randomly generate 100,000 input vectors for VECBEE [8] to ensure both accuracy and efficiency of the logic error estimation.

Important parameters of our framework are listed as follows. The upper bound of MCTS iterations is 26. For $Split$ step, the minimum number of circuits required for the leaf circuit set to undergo further splitting is set to 20. For UCT in $Selection$ step, the weight of HyperVolume w_{hv} = 0.7, while the weight of average error w_e = 0.3. These two weights are determined based on the optimization results of multiple tests. The Margin used in the pair-wise loss $\mathcal{L}_{rank}$ of the Rank-Transformer is 20.

Since our framework enables fast multi-objective optimization, three indicators are used to evaluate its performance, including **delay ratio** (the critical path delay of the best approximate circuit over the accurate one), **area ratio** (the area of the best approximate circuit over the accurate one) and **runtime**. Considering the randomness of logic simulation, our framework performs five optimization runs for each test circuit. The average values of the above three indicators are used as the final evaluation criteria.

B. Multi-objective Optimization Results

We compare our framework with the sate-of-the-art delay-driven and area-driven works under ER and NMED constraints. For delay-driven works, we choose three methods, including: HEDALS [1] using critical error graph, TCAD24 [2] utilizing reinforcement learning, and DCGWO [3] optimizing both critical path depth and area. For area-driven works, VECBEE-SASIMI [8], which combines a greedy algorithm with efficient error estimation, is selected.

Results under ER constraint. We compare the multi-objective optimization performance of our framework with other works under a 3% ER constraint. According to the results presented in TABLE II, our framework obtains greater delay reduction in the majority of cases and reduces more area across all cases. On average, it achieves a 29.84% delay reduction while saving 23.24% in area.

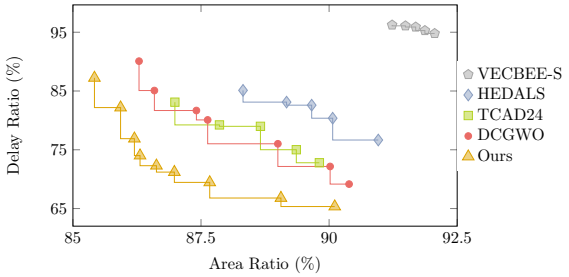


Fig. 4 Multi-objective optimization results across all works on 128-bit multiplier under 1.96% NMED constraint.

Furthermore, we present the optimization results of all works under different ER constraints (1% - 5%), including the average delay ratio trends shown in Fig. 5a and the average area ratio trends illustrated in Fig. 5c. The results indicate that our framework consistently achieves higher levels of optimizations in both delay and area while meeting various error rate requirements.

Results under NMED constraint. We compare the multi-objective optimization performance of our framework with other works under a 1.96% NMED constraint. The results presented in TABLE II indicate that our framework obtains greater delay and area reductions in the majority of cases. On average, our framework achieves a 25.51% delay reduction while saving 15.93% in area, both outperforming other works. The variations in delay and area reductions achieved by all works under different NMED constraints (0.48% - 2.44%) are shown in Fig. 5b and Fig. 5d, respectively. The results also demonstrate that our framework consistently achieves deeper optimizations in both delay and area while meeting diverse error distance requirements.

Additionally, to determine whether the approximate circuits generated by our framework are fully dominant in the objective function space, we evaluate the optimization results of all works on 128-bit multiplier under the 1.96% NMED constraint. We retain all approximate circuits in the 0-th Pareto level of the root circuit set to construct the Pareto frontier of our framework. Based on the results in Fig. 4, the Pareto front circuits generated by our framework still remain on the Pareto frontier among the circuits produced by all works, providing strong evidence of the superiority of our work in multi-objective optimization.

C. Runtime Comparison

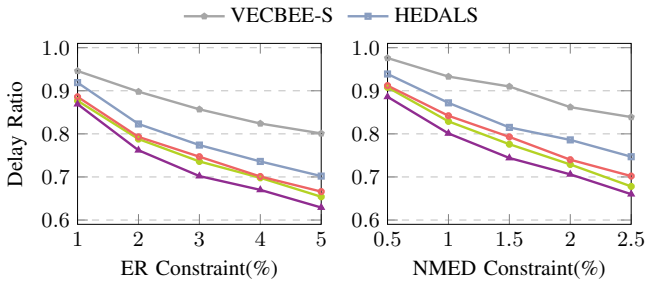
The overall runtimes for the circuit optimization flow under NMED constraint across different methods are listed in TABLE IV. Note that the overall runtime accounts for both the time consumed in approximate optimization and all evaluation processes. The comparison results demonstrate that our framework achieves an average speedup of 1.19× to 4.16× compared to TCAD24 [2]. As shown in Fig. 6, for both DCGWO [3] and our framework, which utilize Synopsys PrimeTime [13] for accurate timing analysis, the primary time consumption in each iteration is attributed to timing analysis and logic simulation. Benefiting from the rank-based partition of the approximate circuit set, the application of multiple LACs to promising circuits in each iteration, and the inherent parallelism of MCTS, our framework achieves rapid search and convergence speeds. Therefore, our framework can achieve fast and efficient optimization.

TABLE II Comparison of multi-objective optimization performance between our framework and other works under the 3% ER constraints.

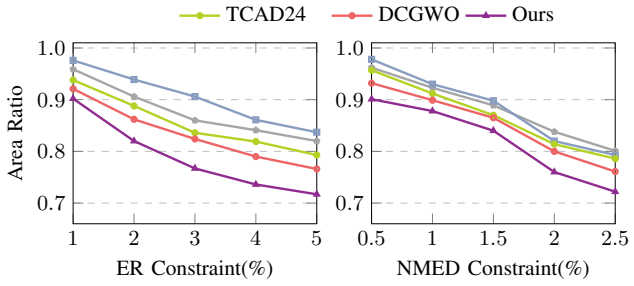
Circuit	VECBEE-S [8]		HEDALS [1]		TCAD24 [2]		DCGWO [3]		Ours	
	delay ratio	area ratio	delay ratio	area ratio	delay ratio	area ratio	delay ratio	area ratio	delay ratio	area ratio
c880	92.16%	86.75%	93.02%	89.22%	83.21%	84.98%	85.21%	82.76%	**77.09%**	**67.54%**
c1908	84.21%	63.37%	**46.12%**	59.36%	48.15%	62.34%	43.97%	57.02%	49.89%	**45.72%**
c2670	79.14%	69.78%	79.64%	94.17%	75.39%	61.28%	76.92%	60.33%	**74.11%**	**56.96%**
c3540	97.93%	94.72%	89.97%	92.46%	84.32%	90.55%	90.61%	87.14%	**75.06%**	**85.18%**
c5315	93.57%	96.68%	94.24%	97.81%	88.55%	90.29%	89.72%	91.12%	**87.06%**	**89.87%**
c7552	91.79%	95.66%	78.53%	99.72%	77.58%	95.34%	79.88%	94.29%	**71.43%**	**91.06%**
cavlc	93.20%	83.78%	96.83%	92.85%	94.28%	85.38%	**92.07%**	89.62%	94.35%	**81.18%**
priority	53.17%	97.16%	47.96%	98.77%	37.28%	98.54%	39.12%	97.22%	**32.25%**	**96.57%**
Average	85.65%	85.99%	77.41%	90.55%	73.60%	83.59%	74.69%	82.44%	**70.16%**	**76.76%**

TABLE III Comparison of multi-objective optimization performance between our framework and other works under 1.96% $NMED$ constraints.

Circuit	VECBEE-S [8]		HEDALS [1]		TCAD24 [2]		DCGWO [3]		Ours	
	delay ratio	area ratio	delay ratio	area ratio	delay ratio	area ratio	delay ratio	area ratio	delay ratio	area ratio
c6288	97.33%	90.81%	73.28%	89.90%	74.38%	91.25%	76.92%	89.01%	**71.95%**	**87.72%**
adder	82.62%	96.42%	78.14%	94.02%	66.88%	92.96%	74.23%	93.69%	**59.23%**	**92.50%**
barshift	90.01%	83.87%	87.46%	89.98%	83.12%	80.55%	82.78%	83.26%	**82.16%**	**75.62%**
max	92.55%	86.44%	81.96%	93.24%	75.99%	85.64%	76.81%	91.80%	**74.86%**	**82.33%**
mult	95.89%	91.79%	82.59%	89.66%	78.99%	88.66%	80.08%	87.63%	**71.20%**	**86.98%**
sine	94.10%	90.28%	91.11%	93.48%	86.14%	88.56%	**82.10%**	89.78%	87.60%	**86.25%**
sqrt	83.23%	91.07%	75.03%	92.10%	75.66%	90.54%	79.16%	**86.29%**	**71.21%**	89.32%
square	92.53%	80.81%	82.13%	76.27%	79.36%	77.54%	82.58%	72.23%	**77.72%**	**71.86%**
Average	91.03%	88.94%	81.46%	89.83%	77.56%	86.95%	79.33%	86.46%	**74.49%**	**84.07%**



(a) Delay Ratios under ER (b) Delay Ratios under NMED

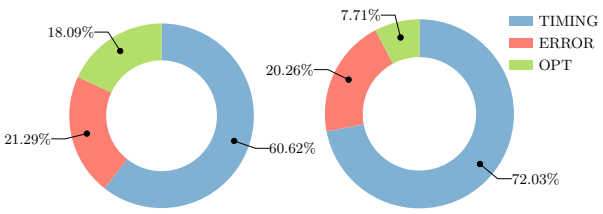


(c) Area Ratios under ER (d) Area Ratios under NMED

Fig. 5 Aveage delay ratios and average area ratios achieved by our framework and others under different ER and NMED constraints.

TABLE IV Overall Runtime (min.) Comparison.

Circuit	VECBEE-S	HEDALS	TCAD24	DCGWO	Ours
c6288	58.60	34.18	22.95	28.05	**20.32**
adder	22.63	18.17	17.68	15.92	**10.09**
barshift	37.89	30.11	22.69	21.26	**17.70**
max	34.68	39.44	23.31	27.13	**19.22**
mult	293.01	187.82	57.83	64.12	**40.38**
sine	71.88	51.63	51.24	42.35	**39.76**
sqrt	551.95	268.8	105.87	132.67	**98.95**
square	57.22	37.13	23.74	**20.95**	24.78
Average	140.98	83.41	40.66	44.06	**33.90**



(a) DCGWO [3] (b) Our framework

Fig. 6 Runtime breakdown in one iteration.

V. Conclusion

This paper presents a rank-based multi-objective Approximate Logic Synthesis (ALS) framework implemented using Monte Carlo Tree Search (MCTS). The core approach focuses on: (1) utilizing non-dominated circuit ranking to guide MCTS in globally identifying approximate circuits with high optimization potential for further exploration, and (2) leveraging the Rank-Transformer to predict path-domain rankings of local approximate changes (LACs) to select high-quality LACs within critical paths. These selected LACs enable effective optimization of both delay and area. Experimental results demonstrate that the proposed framework efficiently optimizes delay and area under various error constraints with low runtime costs.

Acknowledgment

The project is supported in part by the National Key R&D Program of China (No. 2023YFB4402900), the National Natural Science Foundation of China under Grant 62304197, Research Grants Council of Hong Kong SAR (No. RFS2425-4S02, No. CUHK14211824 and No. CUHK14210723), and the MIND project (MINDXZ202404).

REFERENCES

[1] C. Meng, Z. Zhou, Y. Yao, S. Huang, Y. Chen, and W. Qian, "Hedals: Highly efficient delay-driven approximate logic synthesis," *IEEE TCAD*, vol. 42, no. 11, pp. 3491–3504, 2023.

[2] Y. Ye, T. Chen, Y. Gao, H. Yan, B. Yu, and L. Shi, "Timing-driven technology mapping approximation based on reinforcement learning," *IEEE TCAD*, 2024.

[3] X. Hu, Y. Ye, T. Chen, H. Yan, and B. Yu, "Timing-driven approximate logic synthesis based on double-chase grey wolf optimizer," 2024. [Online]. Available: https://arxiv.org/abs/2411.10990

[4] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "Salsa: Systematic logic synthesis of approximate circuits," in *Proc. DAC*, 2012, pp. 796–801.

[5] Y. Wu and W. Qian, "An efficient method for multi-level approximate logic synthesis under error rate constraint," in *Proc. DAC*, 2016, pp. 1–6.

[6] S. Hashemi, H. Tann, and S. Reda, "Blasys: Approximate logic synthesis using boolean matrix factorization," in *Proc. DAC*, 2018, pp. 1–6.

[7] C. Meng, W. Qian, and A. Mishchenko, "Alsrac: Approximate logic synthesis by resubstitution with approximate care set," in *Proc. DAC*. IEEE, 2020, pp. 1–6.

[8] S. Su, C. Meng, F. Yang, X. Shen, L. Ni, W. Wu, Z. Wu, J. Zhao, and W. Qian, "VECBEE: A versatile efficiency–accuracy configurable batch error estimation method for greedy approximate logic synthesis," *IEEE TCAD*, vol. 41, no. 11, pp. 5085–5099, 2022.

[9] J. Ma, S. Hashemi, and S. Reda, "Approximate logic synthesis using boolean matrix factorization," *IEEE TCAD*, vol. 41, no. 1, pp. 15–28, 2021.

[10] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002.

[11] S. Daulton, M. Balandat, and E. Bakshy, "Differentiable expected hypervolume improvement for parallel multi-objective bayesian optimization," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9851–9864, 2020.

[12] Y. Zhao, L. Wang, K. Yang, T. Zhang, T. Guo, and Y. Tian, "Multi-objective optimization by learning space partitions," *arXiv preprint arXiv:2110.03173*, 2021.

[13] Synopsys, "Primetime user guide," https://www.synopsys.com/cgi-bin/imp/pdfdla/pdfr1.cgi?file=primetime-wp.pdf, 2023.

[14] S. Yan, X. Xiong, A. Arnab, Z. Lu, M. Zhang, C. Sun, and C. Schmid, "Multiview transformers for video recognition," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 3333–3343.

[15] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *Proceedings of the 22nd international conference on Machine learning*, 2005, pp. 89–96.

[16] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the iscas-85 benchmarks: A case study in reverse engineering," *IEEE MDTC*, vol. 16, no. 3, pp. 72–80, 1999.

[17] EPFL, "The epfl combinational benchmark suite," https://www.epfl.ch/labs/lsi/page-102566-en-html/benchmarks/, 2019.

[18] Synopsys, "Design compiler user guide," https://www.synopsys.com/zh-cn/implementation-and-signoff/rtl-synthesis-test/design-compiler-graphical.html, 2023.

[19] A. Mishchenko *et al.*, "ABC: A system for sequential synthesis and verification," 2022.