Power Converter Circuit Design Automation using Parallel Monte Carlo Tree Search

SHAOZE FAN, New Jersey Institute of Technology, USA
SHUN ZHANG, IBM T. J. Watson Research Center, USA
JIANBO LIU, University of Notre Dame, USA
NINGYUAN CAO, University of Notre Dame, USA
XIAOXIAO GUO, Meta Platforms Inc., USA
JING LI*, New Jersey Institute of Technology, USA
XIN ZHANG*, IBM T. J. Watson Research Center, USA

The tidal waves of modern electronic/electrical devices have led to increasing demands for ubiquitous application-specific power converters. A conventional manual design procedure of such power converters is computation- and labor-intensive, which involves selecting and connecting component devices, tuning component-wise parameters and control schemes, and iteratively evaluating and optimizing the design. To automate and speed up this design process, we propose an automatic framework that designs custom power converters from design specifications using Monte Carlo Tree Search. Specifically, the framework embraces the upper-confidence-bound-tree (UCT), a variant of Monte Carlo Tree Search, to automate topology space exploration with circuit design specification-encoded reward signals. Moreover, our UCT-based approach can exploit small offline data via the specially designed default policy and can run in parallel to accelerate topology space exploration. Further, it utilizes a hybrid circuit evaluation strategy to substantially reduce design evaluation costs. Empirically, we demonstrated that our framework could generate energy-efficient circuit topologies for various target voltage conversion ratios. Compared to existing automatic topology optimization strategies, the proposed method is much more computationally efficient — the sequential version can generate topologies with the same quality while being up to 67% faster. The parallelization schemes can further achieve high speedups compared to the sequential version.

CCS Concepts: • **Computing methodologies** → **Search methodologies**; Parallel computing methodologies; • **Hardware** → **Circuit optimization**.

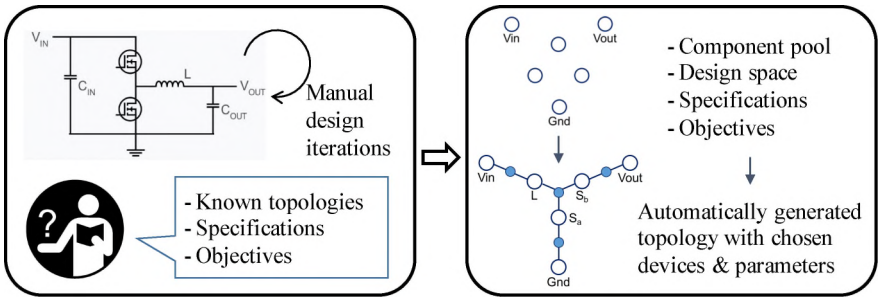
Additional Key Words and Phrases: design automation, circuit synthesis, power converter, circuit topology design, Monte Carlo Tree Search (MCTS), upper-confidence-bound tree (UCT)

ACM Reference Format:
Shaoze Fan, Shun Zhang, Jianbo Liu, Ningyuan Cao, Xiaoxiao Guo, Jing Li, and Xin Zhang. 2022. Power Converter Circuit Design Automation using Parallel Monte Carlo Tree Search . *ACM Trans. Des. Autom. Electron. Syst.* 37, 4, Article 111 (June 2022), 33 pages. https://doi.org/XXXXXXX.XXXXXXX

*Corresponding authors.

Authors' addresses: Shaoze Fan, New Jersey Institute of Technology, University Heights, Newark, NJ, USA, 07102; Shun Zhang, IBM T. J. Watson Research Center, 75 Binney Street, Cambridge, MA, USA, 02142; Jianbo Liu, University of Notre Dame, 275 Fitzpatrick Hall of Engineering, Notre Dame, IN, USA, 46556; Ningyuan Cao, University of Notre Dame, 275 Fitzpatrick Hall of Engineering, Notre Dame, IN, USA, 46556; Xiaoxiao Guo, Meta Platforms Inc., 1 Hacker Way, Menlo Park, CA, USA, 94025; Jing Li, jingli@njit.edu, New Jersey Institute of Technology, University Heights, Newark, NJ, USA, 07102; Xin Zhang, IBM T. J. Watson Research Center, 1101 Kitchawan Rd, Yorktown Heights, NY, USA, 10598.

1084-4309/2022/6-ART111 $15.00
https://doi.org/XXXXXXX.XXXXXXX



Conventional: manual design from known topologies to an optimized design by domain experts

Proposed: design automation framework from specifications to circuit topologies with chosen devices & parameters

Fig. 1. Given a custom power converter design task, the conventional manual approach relies heavily on known topologies and is labor-intensive, computationally expensive, and time-consuming. In contrast, our automatic power converter design framework can explore the design space more effectively, thereby immensely decreasing the development time and cost without compromising the performance.

1 INTRODUCTION

Power converters are ubiquitous in electronic/electrical devices. With the proliferation of customized electrical systems [29], such as electric vehicles, self-powered IoT, wearable/implantable biosensors, the need for custom power converters is rapidly increasing to provide diverse supply power standards. The design specifications, such as voltage conversion ratio, power efficiency, output ripple, form-factor, and cost constraints, differ significantly from application to application. Designing a converter for a specific deployment scenario with certain specifications involves selecting among a large number of components and topologies, configuring the chosen elements, evaluating the design performance via simulations, and iteratively optimizing the system design to yield better performance while satisfying specific resource, technology, and cost constraints. In the conventional manual design process, each of the above-mentioned steps is done manually, causing the whole process to be extremely costly and time-consuming. Moreover, the manual circuit optimization relies heavily on existing circuit topologies, as illustrated in Figure 1. The expensive design process has dramatically hindered the development of novel power converters for fast-paced and innovative custom designs. Hence, there is a pressing need for an automatic circuit design framework that can efficiently search and generate high-quality power converter topologies from the design specifications.

However, how to automate the power converter topology design remains a challenging task. Firstly, topology generation for electronic circuits or integrated circuits (IC) lacks thorough investigation. The state-of-art analog/mixed-signal (AMS) IC design automation mainly addresses device sizing or parameter optimization for a fixed circuit structure [17, 20, 24, 35, 37, 38, 40, 49]. People have also investigated automation methods to accelerate the physical implementation of AMS ICs when schematic/topology design is already done [5, 7, 15, 16, 22]. More recently, researchers have started looking into circuit synthesis [31, 43, 47]. But some of them require substantial domain knowledge, which greatly hindered their generality. Others explore the enormous topology space via exhaustive search, metaheuristic search, or gradient descent, which may not be as efficacious in the non-continuous topology space as for other design tasks.

Fundamentally, automated topology generation is inherently difficult, as it faces challenges due to the immense search space and severe data discontinuity. In fact, the search space increases exponentially with circuit complexity, and the enormous number of possible topologies prohibits

exhaustive or random search. Further, metaheuristic search strategies may get stuck in a restricted set of topologies and thus output sub-optimal results unless the number of random samples becomes large enough. Moreover, unlike device parameters such as transistor width and length, a small shift in the component connections of one topology will very likely lead to significant changes in the circuit performance. As such, search algorithms or optimization methods that rely on continuity between "similar topologies", such as genetic search or gradient descent, may become less effective in reducing search efforts.

Finally, it is time-consuming to evaluate the performance of generated topologies properly. For power converters, this is often a more severe problem because they are usually nonlinear and dynamically controlled switching circuits, which require long simulations to reach their steady states. The conventional Spice simulation [28] is able to provide high-fidelity evaluation results, but this comes with the cost of a long simulation time to achieve the desired precision and control scheme exploration. The evaluation cost per topology can be as high as minutes, making the topology exploration process prohibitively time-consuming.

To address the above challenges, our previous paper [9] proposed a design automation framework for power converter circuit design and optimization. The main contributions are as follows:

- We propose the first automatic power converter design framework that intelligently explores the power converter topology space and generates high-quality candidate circuits based on custom design specifications. As shown in Figure 1, our framework can efficiently locate well-performing topologies with appropriate control schemes and also has the potential to generate novel topologies under specific design constraints.
- For the first time, our framework applies Monte Carlo Tree Search, more specifically the upper-confidence-bound-tree (UCT) variant, to circuit topology generation. Unlike other methods discussed above, UCT, with proper search tree construction and offline knowledge enhanced improvements, can better tackle the data discontinuity issue, making it uniquely suitable for circuit topology generation. Thus, we construct the UCT structure to sufficiently capture the semantics of topologies to explore the topology space more efficiently. Moreover, this UCT structure is able to exploit offline knowledge, which is obtained from a few suitable topologies with smaller sizes and encapsulated into our specially designed default rollout policy, and further accelerate the topology space exploration.
- As the long-running circuit evaluation is the bottleneck of fast topology exploration, we detect isomorphic topologies and adopt a hybrid circuit evaluation approach. Our framework uses a State-Space Averaging method during the topology space exploration, which reduces the time cost of circuit evaluation by orders of magnitude. The circuit candidates generated by the exploration are validated by a high-fidelity Spice transient simulation to filter out the over-optimistic ones.
- We conduct extensive experiments on 5-component (13-port) power converter design tasks. Evaluation results demonstrate that our proposed automatic framework can produce energy-efficient circuits for varying voltage conversion ratios. Furthermore, compared to baseline strategies (i.e., genetic search and random search algorithms) adapted from other circuit design tasks, our framework can generate constraint-satisfied and highly efficient topologies while needing fewer queries for circuit evaluation. Hence, it is up to 70%, 56%, and 50% faster than the baseline strategies for the experimented buck, boost, and buck-boost converter design tasks, respectively.

This paper extends our previous work to further improve the efficiency, enhance the applicability, and parallelize the topology space exploration of our UCT-based automatic power converter design framework. Specifically, we make the following new contributions:

- We conduct an analysis of the power efficiency, voltage conversion ratio, and specification-related quality of 100K randomly generated circuits consisting of 5 devices to form a better understanding of the design space. The analysis inspires us to incorporate mechanisms to improve the performance of our framework and explain why random search does not work well for circuit design tasks.
- To speed up the topology space exploration of our UCT-based framework, we design and implement two main parallelization schemes for UCT, namely root parallelization and leaf parallelization. We also implement additional mechanisms specially designed for circuit exploration on top of the two schemes to further improve their speedup and efficiency.
- We perform an in-depth examination and comparison between the faster State-Space Averaging method and high-fidelity Spice transient simulation. This investigation explicitly reveals the running time and accuracy trade-offs of different circuit evaluation methods and motivates our hybrid evaluation approach.
- We conduct extensive experiments to validate the effectiveness of the hybrid circuit evaluation approach. Results show that the hybrid use of State-Space Averaging and high-fidelity Spice transient simulation in our UCT-based converter generation framework achieve comparable performance but more than 20 times shorter running time than using only simulation.
- We present our design of the unique circuit encoding that reduces isomorphic topologies. Together with the use of a hash table, it can enormously reduce the number of circuits evaluated by the State-Space Averaging method and Spice transient simulation, which in turn reduces the running time of all topology search algorithms.
- We conduct comprehensive experiments to evaluate the efficacy of different parallelization techniques. The evaluation results show that the root parallelization scheme can achieve 9.7x speedup using 13 CPU cores with little performance loss for our framework using the hybrid evaluation approach. Similarly, it can achieve 12.8x speedup using 16 cores with little performance loss for our framework using only the high-fidelity simulation.

Organization. We organize the remainder of the paper as follows. In Section 2, we briefly discuss related works on circuit design automation. Section 3 formally describes the problem statement considered in this work. We introduce our UCT-based power converter design framework in 4. To further enhance the practicality of our framework, we apply parallelization techniques to our UCT design and adopt several mechanisms for reducing the circuit evaluation times, which are discussed in Sections 5 and 6, respectively. Section 7 provides experimental evidence of the effectiveness of our proposed framework on different power converter design tasks with practical design specifications. Finally, we draw conclusions in Section 8.

2 RELATED WORKS

This section discusses the most relevant works on circuit design automation and Monte Carlo Tree-Search (MCTS).

2.1 Circuits Design Automation

With the demands of custom electronics, application-specific design automation of analog/mixed-signal (AMS) and radio-frequency (RF) power management circuits starts to play vital roles in accelerating high-quality electronic circuit designs. However, the traditional manual design routines are inherently time-consuming and rely heavily on domain expertise. To reduce the cost and improve the design quality, mainstream research about circuit design automation are three folded: (1) automating the device parameter optimization for known circuit topologies; (2) automating the

physical implementation for known circuit topologies and parameters; (3) automating the circuit synthesis that directly generates topologies.

Parameter Optimization. Great efforts have been devoted to automating the parameter optimization for predetermined topologies. For example, [1] proposed a random region covering method that can reduce the probability of generating sub-optimal results , [17] proposed geometric-programming-based optimization, [35, 38] used regression and convex/polynomial optimization, and [24] applied a Bayesian optimizer, [20, 40] both adopted model-based reinforcement learning to find the optimal device parameter combinations for analog circuits. Additionally, [37] encoded circuits using graph convolutional neural networks to transfer the parameter optimization knowledge learned between two topologies or between technology nodes of the same circuit. As the circuit evaluation is very time-consuming, Zhang et al. proposed an efficient asynchronous batch Bayesian optimization approach for parallelizing circuit sizing and developed a new acquisition function to better explore the design space of analog circuit synthesis [44].

Physical Implementation. The physical implementation automation for integrated circuits (IC), such as device placement and routing, also plays a vital role in many high-performance AMS/RF circuits. For example, analog generators were proposed in [5, 15], which directly build analog circuit layouts. [22] applied a data-driven approach to check layout symmetry, which is crucial in high-quality AMS physical layout. A feed-forward equalization transmitter layout generator was introduced in [16], which significantly reduced layout time. [7] presented a novel detailed routing framework for AMS layout synthesis to address the sensitive net coupling issues. In [48], the effectiveness of slicing and nonslicing representation in handling placement constraints is investigated. Additionally, the technique of congestion-based virtual sizing is proposed. [41] reduces routing congestion during the placement stage by applying the integer linear programming to formulate the problem of conflicts between multiple congested regions and performing local improvement according to the solution of the integer linear programming.

Topology Optimization. In contrast to parameter optimization and physical implementation that target fixed topologies, recent works have started investigating circuit topology optimization. Specifically, [43] proposed a bi-directional graph neural network (GNN) model that learns to simulate the electromagnetic properties of distributed circuits. Via back-propagating the gradient, this GNN model can also be used to optimize the circuit parameters and topology. However, due to the special electromagnetic property of coupling decays in distributed circuits, an "edge" in a circuit topology is determined by the physical distance between two nodes and its impact on the circuit is continuous and decomposable. In comparison, the edge in a power converter topology is determined by whether the two components are connected in the circuit, and removing one edge may utterly change the performance of the circuit (e.g., from valid to invalid). Hence, the gradient back-propagation approach with the GNN model in [43] cannot be directly applied to the power converter design task. For topology synthesis for large analog integrated circuits, [47] presented a graph-grammar-based circuit topology representation, which hierarchically decomposes a circuit until reaching the basic predefined building block. To reduce the search space, this work focused on designing meaningful decomposition rules and circuit formation rules that domain experts manually add while the generation is performed using an *exhaustive search* within the representation space. Graeb et al. [11] proposed the sizing rules resulting from constraints guaranteeing the dedicated function and robustness for analog CMOS circuit synthesis using a hierarchical component library. The hierarchical component library designed by domain experts reduces the complexity of the problem so that the *exhaustive search* becomes affordable. Thus, both works require substantial domain knowledge of sub-circuit structure to construct large-scale analog circuits, in order to reduce the search space under *exhaustive search*. Similarly, [25] reduces the search space via a

set of expert-specified hierarchically-organized analog building blocks, but it uses *genetic search* instead. For searching in the circuit topology space without expert-specified building blocks, [31] developed a *genetic search* algorithm, where the device types of components in the topology are essentially fixed. We extend it to allow changing component types and compare it with our proposed framework.

AI-assisted Power Electronic Design. The advancements of artificial intelligence (AI) and IC design automation have introduced marvelous opportunities for power electronic circuit design automation [46]. Existing works have been looking into methods that greatly reduce circuit evaluation time by advancing surrogate models that approximate the system dynamics (e.g., electromagnetic properties, thermal characterization, and wire costs) with lower computational efforts [23, 39, 43, 45]. Researchers have also applied AI techniques and shown some successes in modeling and optimizing other aspects of power electronic systems, such as component model, system parameters, and post-layout performance [14, 33, 37, 49]. While the aforementioned efforts in power converter design automation address parameter optimization and modeling in power electronics, there has been little work that investigates efficient topology synthesis with minimal prior knowledge and human intervention.

2.2 Monte Carlo Tree Search

For problems that can be formulated as Markov decision processes, Monte Carlo Tree Search (MCTS) is a widely used search and planning framework that learns a value function and finds the optimal decisions via sampling-based search. It has been applied to various applications [3] and continuously improved since being proposed. For example, [8] defines a general backup operator, provides fine-grained control of the tree growth of MCTS, and allows efficient selectivity methods. The upper-confidence-bound-tree (UCT) was proposed in [18]. UCT applies bandit ideas to guide the Monte-Carlo planning and significantly outperforms other alternatives. [10] further improves the searching efficiency of UCT by designing three techniques to combine online and offline knowledge. The MCTS method has also been used to provide training data for a deep-learning model used as a real-time Atari game playing agent [13].

Due to the popularity of MCTS, parallelization techniques have also been applied to speed up the execution time of MCTS despite the challenges [32, 42]. In particular, [6] proposed the basic parallelization approaches of MCTS, including leaf parallelization, root parallelization, and tree parallelization. These parallelization strategies have shown good performance not only in the Go game [12, 27, 34] but also in many other applications [2, 4, 19, 21]. For example, the parallelization mechanism proposed in [21] achieves near-linear speedup with limited performance loss across many different Atari Games. The performance of different parallelization strategies and the most suitable improvement techniques depend highly on the specific application, the implementation language, and the computing platform.

3 PROBLEM STATEMENT

We investigate the automatic power converter design problem with topology generation, device type selection, and control parameter tuning. This section first describes the custom design specifications of power converters considered in this work, followed by their topological representation and parameters. Next, we formulate the problem as an optimization problem with encoded custom design specifications.

Custom Design Specifications. We consider two primary design metrics of power converters, namely, *voltage conversion ratio* γ and *power conversion efficiency* η, as shown in Figure 2. The voltage conversion ratio is the ratio of the output voltage to the input voltage, i.e., $\gamma = V_{\text{out}}/V_{\text{in}}$,

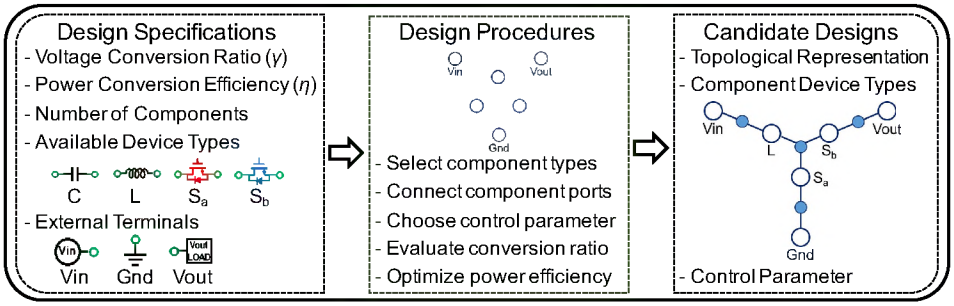


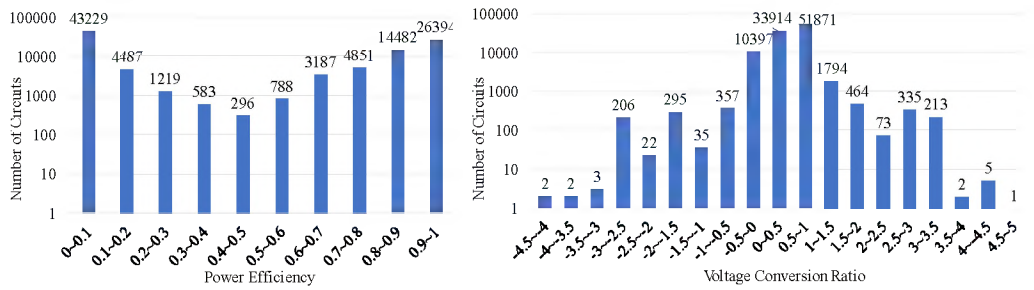
Fig. 2. Custom power converter design task from design specifications to candidate designs.

and is the main constraint for the generated power converter. Generating power converter designs with higher power conversion efficiencies is the optimization goal of the custom design task. Other constraints include the number of components in the converter topology and the types of available devices. The design task evaluated in this work only considers devices including capacitors C, inductors L, phase-I switches S_a, and phase-II switches S_b, but it can be easily extended to other device types.

Topological Representation and Parameters. A candidate power converter design consists of a *topological representation* s and a *switching control parameter* d. Specifically, the topological representation contains a set of components with ports and edges connecting the ports. Each component has a device type (from the set of available devices) with device parameters (e.g., inductance, capacitance, and transistor dimensions) and two ports (i.e., left and right ports). Note that each component is nondirectional — switching all the connections of its two ports results in the same circuit, despite needing different indexes to distinguish the two ports in the topological representation. Additionally, there are three *external terminal ports*: the input voltage port `Vin`, the output voltage port `Vout`, and the ground port `Gnd`. The edges in the topological representation specify the connections between the components' ports and the terminal ports. The switching control parameter specifies the duty cycle of a candidate design, which often affects the output voltage. In this work, the design task involves designing the device types of components, the edges connecting ports, and the control parameter, while the device parameters for each device type are predefined. Thus, this work focuses mainly on the challenging topology design problem and leaves the integration with existing device parameter optimization methods as future work.

Circuit Evaluation. After a circuit is generated, it is evaluated to determine its performance towards required specifications. Given a generated power converter circuit with topological representation s and control parameter d, the Spice-based transient simulation is conducted to obtain the voltage conversion ratio $\gamma_{s,d}$ and power efficiency $\eta_{s,d}$.

To better understand the design space, we performed an in-depth analysis on the power conversion efficiency and voltage conversion ratio distributions of randomly generated circuits with five components shown in Figure 3(a) and Figure 3(b), respectively. We can observe that most randomly generated circuits reside either in the low-efficiency range (0-10%) or the high-efficiency range (80-100%). Note that Figure 3(a) is shown in log-scale, so more than 80% randomly generated circuits have very low or very high efficiencies. Furthermore, among the circuits with high efficiencies, most of them have a voltage conversion ratio approaching 1. These are the circuits that have direct or indirect shortcut connections between `Vin` and `Vout`, which cannot be used for power converters. These observations reveal both the challenges of finding high-efficiency circuit candidates and the ineffectiveness of random search in the design space.



(a) Power efficiency distribution (b) Voltage conversion ratio distribution

Fig. 3. Distributions of the part of the randomly generated circuits with five components with respect to power conversion efficiencies and voltage conversion ratios. Invalid circuits, e.g., circuits with unconnected components or ports, have been removed from the random generation.

Circuit Generation Objective. Given the custom design task with a target voltage conversion ratio γ_0, the goal of our framework is to automatically generate the topological representation s (with chosen component types and edges) and configure the control parameter d of the power converter circuit. Specifically, the objective can be described as:

$$s^*, d^* = \arg\max_{s \in S, d \in D} [U_{\gamma_0}(\gamma_{s,d}, \eta_{s,d})], \tag{1}$$

where S and D denote the set of topological representations and the set of control parameter configurations. U_{γ_0} is a utility function that evaluates a circuit design's conversion ratio and efficiency for the custom design task with target conversion ratio γ_0. Specifically, the utility function is formulated as:

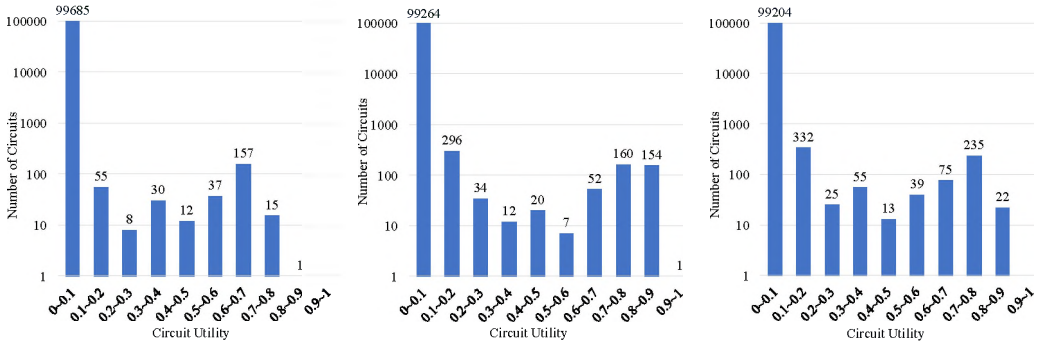
$$U_{\gamma_0}(\gamma_{s,d}, \eta_{s,d}) = \eta_{s,d} \cdot \delta(\gamma_{s,d}, \gamma_0). \tag{2}$$

In our formulation, δ measures how close the obtained conversion ratio $\gamma_{s,d}$ and the target conversion ratio γ_0 are. In our experiments, we use $\delta(\gamma, \gamma_0) = 1.1^{-\left(\frac{15(\gamma-\gamma_0)}{|\gamma_0|}\right)^2}$. The utility function is set to 0 when the topology is invalid or incomplete. Under this formulation, a circuit has a higher utility if it has a higher efficiency and a conversion ratio closer to the target ratio.

Analysis of Design Space. We plot the utility distributions of the randomly generated circuits above given different target power conversion ratios in Figure 4. Note that Figure 4 is also shown in log-scale. The percentage of the high utility buck-boost, buck, and boost circuits (utility larger than 0.6) is only 0.173%, 0.315% and 0.332%, respectively. This analysis confirms that the number of well-performing circuits given specific design specifications is very small. Essentially, given the component pool and the possible connections between them, there are combinatorially-many possible circuits. Among these circuits, most have low power efficiencies or undesirable conversion ratios. Thus, it is computationally intractable to search over all possible circuits to find the optimal one, and it is also inefficient to find high-quality circuits via random search.

We also manually look into the randomly generated circuits to understand topology properties that affect the performance of circuits. By carefully looking into bad topology candidates (i.e., efficiency close to zero or conversion ratio close to one), we find that there are similar paths associated with such topologies. In summary, the topologies with bad performance mostly share one of the following features:

- Pure inductive paths between two of the terminal ports `VIN`, `VOUT`, or `GND`;



(a) Target conversion ratio -2 (b) Target conversion ratio 0.5 (c) Target conversion ratio 2

Fig. 4. Utility distributions of randomly generated circuits with five components under different target power conversion ratios. Conversion ratios -2, 0.5, and 2 correspond to buck-boost, buck, and boost converters, respectively.

- No path between VIN and VOUT that does not by-pass GND.

These findings are consistent with our domain knowledge of power electronics. Thus, we define those paths as prohibited paths and do not perform circuit evaluation for circuits containing prohibited paths. Additionally, we combine such knowledge into our UCT-based framework via action pruning, which will be described in Section 4.3.

The topology design space in this work is comprehensive, except that we removed isomorphic circuits and circuits violating human-designed rules. For the parameter design space, we have selected specific values for the optimized precision-cost trade-off. We identify the control parameter, namely duty cycles, as the first-order variable that significantly alters the voltage conversion ratio and power efficiency. As such, we optimize the duty-cycle parameter. On the other hand, to reduce the evaluation cost per topology, we fix the second-order parameters in this work, such as capacitor capacitance, inductor inductance, and source resistance. In principle, our proposed framework can be extended to also optimize other parameters. Alternatively, once a custom topology is reached, further parametric optimization is feasible through state-of-the-art parameter optimization methodologies discussed in Section 2. Finally, after the topologies and parameters are generated, fine-grained analyses, such as LC-coupling and PVT variation, are needed to validate their performance in practice, which is beyond the scope of this work.

4 UPPER-CONFIDENCE-BOUND-TREE-BASED CIRCUIT GENERATION

In this work, we propose an automatic power converter design framework that utilizes the upper-confidence-bound-tree (UCT) variant of Monte Carlo Tree Search , together with several efficiency-enhancing mechanisms, to intelligently explore the topological representation space and locate candidate designs with high utility scores. This section describes our formulation and main algorithm design of the topology generation using upper-bound-confidence-tree (UCT). To speed up the topology exploration, we investigate several parallelization techniques for the UCT-based topology design, which will be discussed in Section 5. Additional mechanisms to reduce the circuit evaluation costs will be presented in Section 6.

We formulate the circuit generation task as a sequential decision-making problem. Instead of synthesizing the entire topology all at once, the component device types and connections between ports are added step by step, as illustrated in Figure 5. At each step, it decides which device to add to

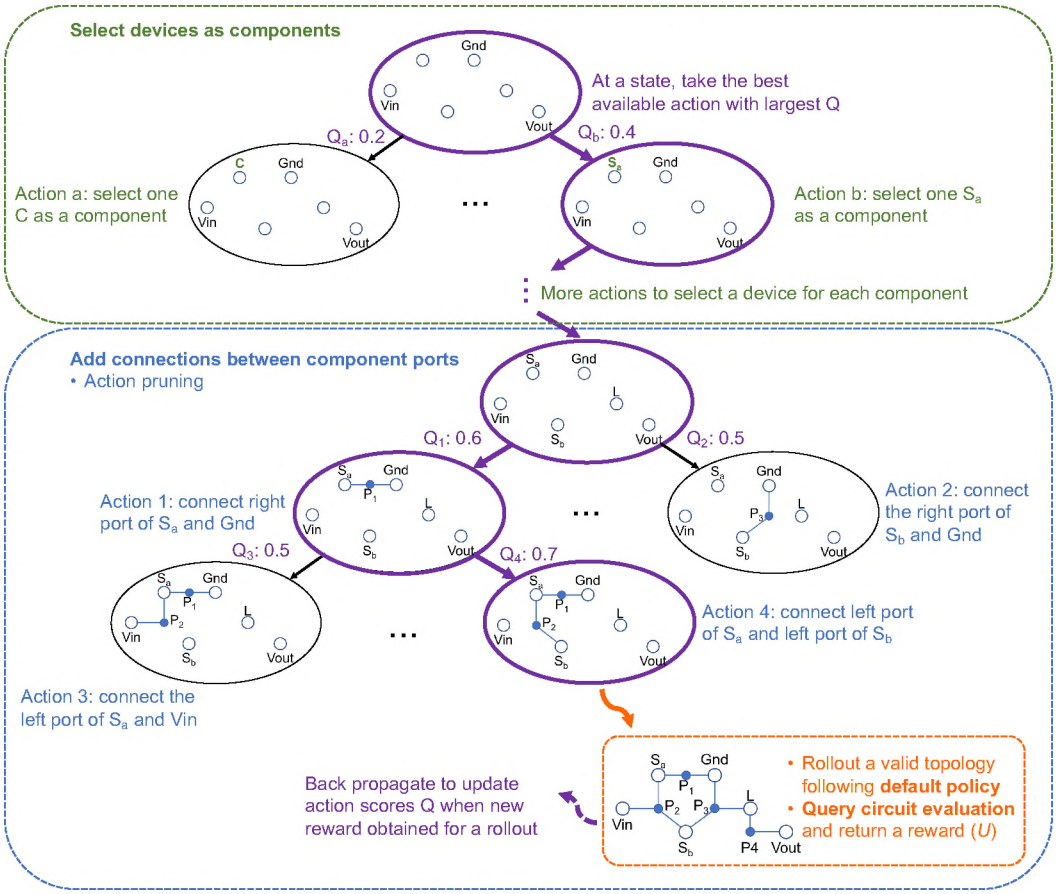


Fig. 5. UCT-based Topology Generation Overview: The generation starts with a topology having empty components and 3 terminals. It first makes device selections (shown in green) and then connects ports of components and terminals (shown in blue), where the actions are specially designed to reduce isomorphism and invalid circuits. The look-ahead tree construction and action score calculation (shown in purple) follow the UCT algorithm, which utilizes upper confidence bound in exploitation and exploration trade-offs. To improve the sample efficiency and rollout speed, we designed a default policy and adopted a fast evaluation technique (shown in orange).

the existing circuit or which pair of component ports to connect in the circuit. Under this multi-step formulation, we construct the UCT-based algorithm to build look-ahead trees to estimate the design choices, such as selecting devices and connecting device ports. In tree building, UCT utilizes upper confidence bound for effective exploitation and exploration trade-offs. Our multi-step formulation and UCT-based algorithm allow physics-aware connection pruning and removes many isomorphic topologies by construction. We further improve the topology generation by incorporating offline knowledge from the pre-collected dataset through a default policy.

4.1 Sequential Circuit Topology Generation

Formally, we formulate the power converter circuit topology generation as a sequential decision task. More specifically, we model the topology generation as a Markov Decision Process, namely

a 4-tuple of $\langle S, A, T, R \rangle$, representing the state set S, action set A, state transition function T, and reward function R. As shown in Figure 5, the topology generation always starts with an empty topology of the power converter and has two phases with multiple steps: the device type selection phase and connection selection phase.

In the t-th step, the state $s_t \in S$ is a partial or complete topology of the power converter circuit. Inspired by the simple fact that circuit topology is a graph, each state s_t maintains a component set, a port set, and an adjacency matrix specifying the connections between each pair of ports. The action set A_t depends on the current state s_t. For a state in the device type selection phase, an action $a_t \in A_t$ decides whether a device type is selected for a component. For the connection selection phase, an action either decides to skip adding more connections or decides which port is connected to the port under consideration. Given our state and action formulation, the state transition is a deterministic function $s_{t+1} = T(s_t, a_t)$, which maps the current partial topology and the action choice to the next topology. After transiting to an new state s_{t+1}, the action set also need to be updated to A_{t+1} according to the new state s_{t+1}.

The reward function R_{γ_0} encodes the custom design objective U_{γ_0} in Equation 2 such that

$$R_{\gamma_0}(s_t, a_t) = \max_{d} U_{\gamma_0}(\gamma_{s_{t+1},d}, \eta_{s_{t+1},d}) - \max_{d'} U_{\gamma_0}(\gamma_{s_t,d'}, \eta_{s_t,d'})$$

The reward function computes the difference in utility between the previous state s_t and the new state s_{t+1}, each with optimized control parameters d and d'. The difference-based reward formulation ensures our sequential circuit topology generation formulation includes the optimal solution to the custom design task. We directly use the utility function in equation 2 in the reward function to represent the quality of a state. For states representing valid and complete circuits topologies, the utility is between 0 and 1. For states representing invalid or partial circuits topologies, the utility function returns 0.

The optimization objective is to find an action sequence that maximizes the reward of the final topology:

$$a^*_{0:T} = \arg\max_{a_{0:T}} \sum_{t=0}^{T} R_{\gamma_0}(s_t, a_t)$$

Remark. Note that the above objective requires computing the maximum utility over the different control parameters. Sweeping all control parameters to find the one with maximum utility is not the only way for the formulation. Alternatively, the sequential decision task can be formulated to have one additional phase with actions for choosing the control parameter (and device parameter if needed). Then, the optimization objective becomes finding an action sequence that maximizes the reward of the final circuit with a specific topology and chosen parameters. Our proposed framework works for both formulations. We implement both versions and conduct experiments to compare their performance. In terms of the quality of the generated circuits, both versions have similar performance. Because the fast circuit evaluation method presented in Section 6.2 can compute the maximum utility over the different control parameters faster, this formulation gives some advantage to our framework using the fast evaluation method in terms of total running time.

4.2 Upper-Confidence-Bound-Tree-based Topology Generation

To effectively solve the above sequential decision-making problem, we adopt Monte Carlo Tree Search, which has shown success in many domains, like playing the Go game [34] and solving the parking problems in Tesla's Autopilot software [36]. Monte Carlo Tree Search has been proven to work well for large state-space sequential decision-making optimization problems, which suits the large topological representation space in our problem formulation. Among different variants of Monte Carlo Tree Search algorithms, the upper-confidence-bound-tree (UCT) algorithm is the most

popular one that uses the Upper Confidence Bound to well balance the exploration and exploitation trade-offs.

At a high level, UCT builds a look-ahead tree and greedily selects an action based on the actions' estimated scores at each step. In our proposed framework shown in figure 5, each tree-node of UCT corresponds to a partial or complete topology (i.e., a state) and each tree-edge corresponds to a device type or connection selection (i.e., an action).

To accurately estimate the action scores Q, UCT performs multiple *look-ahead rollouts* (also called simulations in the literature), which sample the remaining action sequence following a *default policy* and evaluate the quality of the samples in terms of reward. UCT is an anytime optimization algorithm with three parameters, the number of look-ahead rollouts, the maximum depth (uniform for each rollout), and an exploration parameter. In general, the larger the number of rollouts and the depth parameter are, the slower UCT is, but the better it is. In our UCT-based algorithm, a rollout returns the reward when reaching a terminal state, instead of a fixed maximum depth like in conventional UCT design. A state is called a terminal state if its corresponding topology is complete or if there is no valid action that can complete the topology. In this way, the rollouts are more likely to find valid circuits with positive rewards.

Compared to other Monte Carlo Tree Search algorithms, UCT utilizes the Upper Confidence Bound in the tree building process to provide improved estimations of action scores. UCT computes a score for each action a_t at a state s_t as the sum of the exploitation and exploration terms, as follows:

$$Q^{\mathrm{UCT}}(s_t, a_t) = Q^{\mathrm{MC}}(s_t, a_t) + \sqrt{\frac{\log(n(s_t))}{n(s_t, a_t)}}$$

where Q^{MC} is an exploitation term and the second one is an exploration term.

The Q^{MC} is the Monte Carlo average of the sum of rewards obtained from the look-ahead rollouts

$$Q^{\mathrm{MC}}(s_t, a_t) = \frac{1}{M} \sum_{m=1}^{M} \sum_{t'=t}^{T} R(s_{t'}^m, a_{t'}^m)$$

where m identifies a look-ahead rollout in the total M rollouts.

The exploration term $\sqrt{\log(n(s_t))/n(s_t, a_t)}$ is the Upper Confidence Bound, where $n(s_t)$ is the number of visits for the state node s_t and $n(s_t, a_t)$ the number of visits of the action a_t at state s_t.

UCT selects the action to rollout greedily with respect to this summed score using the look-ahead tree. Once the input-parameter number of rollouts are produced each to the maximum depth, UCT returns the exploitation term for each action at the root node.

4.3 Combining Knowledge into UCT via Action Pruning

As discussed above, UCT estimates a state-action pair's score $Q^{\mathrm{MC}}(s_t, a_t)$ more accurately when more rollouts are performed and the total number of rollouts is fixed according to the affordable computation cost. Hence, pruning the action space can improve UCT's search efficiency. Therefore, we design several physics-informed action pruning strategies to reduce the actions that lead to invalid or isomorphic topologies.

Reduce Invalid Topologies. Based on the knowledge of electronic circuits, we pose a set of constraints when adding connections to avoid generating invalid power converter circuits. Specifically, we do not include an action a of connecting two ports into the candidate action set A if adding this connection leads to one of the following situations:

- a shortcut;
- a direct connection between terminal ports `VIN`, `VOUT`, or `GND`;

- a prohibited path;
- a disconnected circuit.

Here, the prohibited paths are the paths that violate basic circuit principles. For example, if we connect VIN and GND only with an inductor, it is equivalent to a power shortcut. A circuit is considered disconnected if there is at least one port not connected to any of the terminal ports via any paths of connections after the generation process.

Together with the action pruning rules described below for connection selection, we can identify such disconnected topologies early in the generation process. Specifically, if there is no more allowed connection that can be added to a port and there is no path connecting this port to any of the terminal ports, then this partial topology will eventually become a disconnected circuit even if additional connections are selected between other ports. Following this rule, if we detect a state that represents a partial topology that will generate unconnected topologies, we no longer expand the state at this point. To do so, we mark the corresponding state as a terminal state with a reward of 0, so no more actions can be taken from this state.

Reduce Isomorphic Topologies. Combining the states representing isomorphic topologies can also improve UCT's search efficiency, since the rewards of rollouts from all these states can be collectively used to estimate the score more accurately. Hence, we propose the following methods to reduce the generated isomorphic topologies by construction.

First, we split the device type selection phase into multiple rounds, where each round has an ordered set of available device types, e.g., {S_a, C, L} in the second round. The set in the first round includes all device types. The ordering of device types is set to S_a, S_b, C, and L. In each step of a round, each available device type is considered for selection, e.g., selecting one S_a for a component or skipping adding S_a. If a device type has been skipped in the current round, it will be removed from the set of available device types for the next round. For instance, if we have added one S_a and one L but skipped C in the second round, the set for the third round becomes {S_a, L}. The device type selection phase ends when the number of selected devices is equal to the number of components. In this way, every state in this phase is unique (i.e., representing different device selections), while all combinations of device selections can be generated.

Next, for the connection selection phase, we number all the ports where the terminal ports have the smallest indexes. We consider each port for adding connections with other ports one by one. Although a connection in a converter topology is not directed, we only allow a port with a smaller index to be connected to a port with a larger index. Thus, this connection can only be added once. In addition, since a device is nondirectional, we always have its left port be connected before its right port when both ports have no connection. Following the above rules, many actions that lead to states representing isomorphic topologies are pruned.

4.4 Combining Knowledge into UCT via Default Policy

Due to the large space of topologies, the computation cost for finding a good topology can be high even after action pruning. One reason is because with no prior knowledge available rollouts can only follow a random action selection policy. However, as shown in Figure 4, the probability of finding good circuits from randomly generated ones is quite low. Hence, we further improve the effectiveness of UCT by replacing the random default policy with a data-driven one.

In particular, we randomly generate some topologies with fewer components and evaluate their efficiencies and conversion ratios. For example, we collect a small data set with 3-component topologies for the design task of 5-component topologies. Among them, we find the *good topologies* with higher rewards and collect their device selections and all paths of connections between the terminal ports. Exploiting the information obtained from good topologies with smaller sizes, we

develop the following two default policies for the two topology generation phases: *node selection* for selecting devices according to device combination information, and *edge selection* for adding connections according to the path information.

Node Selection. We collect the number of times a device selection combination C (e.g., $\{S_a, S_a, L\}$) occurring in the good topologies, denoted by $n_d(C)$. This information is encoded into the default policy, such that its probability of a device selection combination C' is approximately proportional to the collected device selection distribution. Specifically, we calculate the weight of a device selection combination C' as $w(C') = \sum_{C \subseteq C'} n_d(C)$. The total weight is defined as $W_d = \sum_{C''} w(C'')$. Then the probability of the device selection combination C' generated by the default policy is

$$\mathbb{P}_d(C') = \frac{w(C')/W_d + \epsilon_d}{\sum_{C''}(w(C'')/W_d + \epsilon_d)}$$

where ϵ_d is a small constant that decides by how much the default policy follows the collected distribution. In our experiments, ϵ_d is set to 0.01.

Edge Selection. We also collect the number of occurrences of a *good path* of connections between any two of the terminal ports (VIN, VOUT, and GND) in the good topology data set, denoted as $n_d(p)$. When considering a port e for adding a connection onto the current partial topology s_t, we first construct a set of all possible good paths $P(e, s_t)$. A path in $P(e, s_t)$ must contains port e and adding this path to s_t cannot lead to invalid topologies (e.g., shortcuts). Next, for any connection (e, e') allowed to be added to s_t, we calculate the weight of taking this action as

$$w(e, e') = \sum_{p \in P(e,s_t) \& (e,e') \subseteq p} n_d(p)$$

Similar to node selection, we can calculate the total weight of all allowed actions as $W_p = \sum_{(e,e'')} w(e, e'')$. Then the probability of the connection selection action (e, e') generated by the default policy is

$$\mathbb{P}_p(e, e') = \frac{w(e, e')/W_p + \epsilon_p}{\sum_{(e,e'')}(w(e, e'')/W_p + \epsilon_p)}$$

where ϵ_p is a small constant and is set to 0.4 in the evaluation. Note that although the weights are calculated using the good paths in the data set with fewer components, longer paths with more components can be generated with the help of ϵ_p.

With the default policies, we can bias UCT towards searching the topology space that may contain high-reward ones.

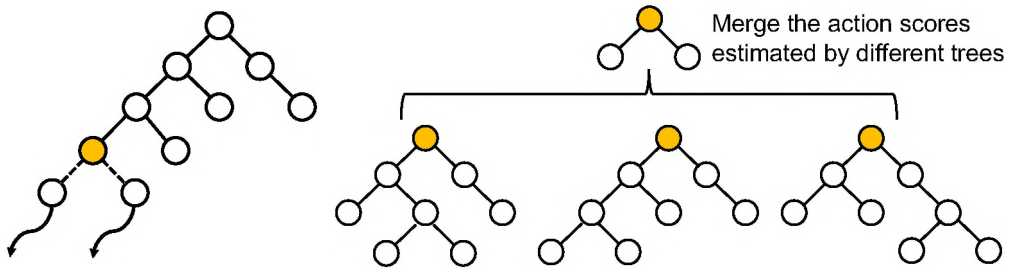
5 PARALLELIZING UCT-BASED CIRCUIT EXPLORATION

In this section, we develop parallelization techniques to speed up the circuit exploration of our UCT-based framework. We explored two main parallelization schemes for UCT: root parallelization and leaf parallelization. To improve the speed up, we also implemented additional parallelization techniques on top of these two schemes.

5.1 Leaf Parallelization Scheme

We first explored the leaf parallelization scheme for our UCT-based framework. As described in Section 4.2, UCT estimates the action scores Q by performing multiple look-ahead rollouts. One natural idea for parallelizing UCT is to perform the multiple rollouts in parallel, as illustrated in Figure 6(a).

Based on this intuition for UCT leaf parallelization, we first generate multiple threads at the start of the execution, where the total number of threads is equal to the number of available cores. Additionally, each parallel thread is pinned to one dedicated core. Whenever we expand a tree-node



(a) Leaf Parallelization (b) Root Parallelization

Fig. 6. Illustration of different UCT parallelization schemes.

during the look-ahead tree building process, multiple threads will work in parallel to perform the rollouts. In particular, each parallel thread will follow the default policy individually to generate one circuit and call the circuit evaluation to obtain the reward of this circuit.

Compared with the original UCT, leaf parallelization can perform more rollouts within the same time, which results in a more precise reward estimation of the nodes in the tree. If there is no limit on the number of queries to the circuit evaluation, leaf parallelization certainly improves the performance. This is simply because more rollouts can be done within the same time to help the action selection. On the other hand, when the number of queries is limited, there is a trade-off between the running time of the UCT-based circuit exploration and its performance.

Additionally, in order to calculate the reward estimation of multiple rollouts, leaf parallelization needs to wait for all parallel threads to return with circuit rewards. However, the circuit evaluation times for different circuits can vary, which is especially evident for the high-fidelity Spice transient simulation. Hence, during the execution, some parallel threads may be blocked by waiting for the slowest thread to return, which may restrict the speedup of leaf parallelization.

5.2 Root Parallelization Scheme

In contrast to sequential UCT and leaf parallelization scheme where only one look-ahead tree is built, the root parallelization scheme builds multiple trees in parallel by multiple threads. Hence, the parallelization naturally comes for building multiple trees. However, if the parallel threads split the prefixed number of rollouts and build their individual trees without any communication, each tree can only observe the rewards of a small fraction of rollouts, resulting in inaccurate estimations of action scores and hence potentially worse generated circuits.

To alleviate this issue, we implement the UCT root parallelization to communicate between multiple trees and collectively select an action at each step. Specifically, in each step, parallel threads start with the same current state, expand their individual trees by performing multiple rollouts, and update the action scores on their own trees. All of these are performed independently in parallel. Once a prefixed number of total rollouts is reached, UCT root parallelization needs to determine the best action to take at the current state and transition to the next state. This decision is made by the parallel threads (trees) collectively, so that a single best action is selected and all threads transition to the same next state, as illustrated in Figure 6(b).

There are two potential ways to select an action from multiple parallel trees: voting and merging. Voting means that each tree votes for an action (according to its individual action score estimations calculated for all valid actions given the current state), and the action with the highest number of votes will be selected. In contrast, merging means that we first merge the action scores from

different trees and then select the action with the highest score. Here, the merge is performed by taking the sum of the Monte Carlo average of rewards of all the look-ahead rollouts performed in different trees.

Formally, let $R_0, R_1, \cdots, R_{m-1}$ denote the roots of the m trees given the current state s_t, where each root R_i has k action children $a_0^i, a_1^i, a_2^i, \cdots, a_{k-1}^i$. When constructing each tree, we record the number of visits $n(s_t, a_j^i)$ and the Q-value $Q^{\mathrm{MC}}(s_t, a_j^i)$ of every action child. To decide the best action to take, we merge the roots of trees into one root R^* with action children a_i^*, where the number of visits $n(s_t, a_i^*)$ and the Q-value $Q(s_t, a_i^*)$ of a_i^* are calculated as follows:

$$n(s_t, a_i^*) = \sum_{j=0}^{m-1} n(s_t, a_j^i)$$

$$Q(s_t, a_i^*) = \frac{\sum_{j=0}^{m-1} n(s_t, a_j^i) Q(s_t, a_j^i)}{n(s_t, a_i^*)}$$

We implemented both action selection methods and conducted some experiments to compare the performance of these two potential methods. Results indicate that merging outperforms voting, which is mainly because the merged action scores can maintain the global information of different trees. Hence, we use merging to select actions for UCT root parallelization.

5.3 Additional Parallelization Technique to Improve the Speedup

In the implementation of our parallel UCT-based framework, we also introduce the following additional techniques to improve its speedup.

Improving Efficiency of UCT Root Parallelization by Prioritizing Trees. A straightforward implementation of UCT root parallelization, which usually works well for other tasks, is to split the total number of rollouts evenly between different parallel trees. However, as also discussed for leaf parallelization, the major part of the execution time spends on circuit evaluation, where the circuit evaluation times for different circuits can vary. Additionally, not all rollouts result in a circuit evaluation. For example, our framework does not query the circuit evaluation for invalid topologies or topologies that have already been evaluated. Therefore, evenly splitting the number of rollouts may not result in the same running time of different threads. Since the parallel threads need to synchronize and communicate to select the action to take at the end of each step, faster running threads have to wait for long-running ones. This, again, may hurt the speedup.

Based on the above observation, we prioritize trees for UCT root parallelization. In particular, instead of fixing each independent tree with the same number of rollouts, we allow the trees to grow dynamically and only require that the total number of rollouts performed by the m trees remains the same as the prefixed value. To satisfy this requirement, we keep track of a global counter c, which counts the number of rollouts performed by all trees. We initialize the counter c equal to the prefixed value, decrease c by 1 when any of the trees expands a node, and stop building the tree when $c = 0$. In this manner, we implicitly prioritize the trees with a shorter circuit evaluation time and automatically achieve better load balancing between the m cores.

Reducing Redundant Circuit Evaluation. For both leaf and root parallelization schemes, different parallel threads may end up evaluating the same circuit generated by different rollouts. Again, since circuit evaluation time is the major part of the execution time of the UCT-based framework, it is critical to reduce redundant circuit evaluation. We achieve this by maintaining a hash table storing all previously evaluated circuits observed by each look-ahead tree and their corresponding rewards. Thus, we only query the circuit evaluation if the circuit does not exist in

the hash table. For root parallelization, each thread independently builds its own tree but all the threads shared a circuit hash table. When a thread finishes evaluating a newly observed circuit, it stores this circuit in the hash table, then the other threads do not need to repeat the evaluation when observing this circuit in its own tree.

Parallelizing the Hybrid Circuit Evaluation. Note that all the above parallelization techniques are applicable to our UCT-based framework with any circuit evaluation approach. In Section 6.2, we will present a hybrid circuit evaluation approach that uses the faster State-Space Averaging method to evaluate the circuits observed during tree building and conducts high-fidelity simulation only for the top circuit topologies with highest utility. Since the high-fidelity simulation is significantly slower than the State-Space Averaging method, the final high-fidelity simulation for top circuits becomes the bottleneck in reducing the total running time when using the hybrid approach. Hence, to speed up the running time, we also parallelize the final simulation by conducting the high-fidelity simulation of the top circuits in parallel by multiple threads. Our implementation maintains a shared circuit queue for threads. After starting the parallel simulation, each thread repeatedly takes a circuit out of the shared queue to conduct high-fidelity simulation until the queue is empty.

6 REDUCING CIRCUIT EVALUATION COSTS

To address the issue of costly circuit evaluation, we design a special circuit encoding that is able to detect many isomorphic circuits in the combinatorial topology space. With the help of this circuit encoding and a hash table, our design automation framework only needs to evaluate the unique circuits generated during the circuit exploration by UCT. Additionally, our design automation framework uses a hybrid circuit evaluation strategy. We use a State-Space Averaging technique to evaluate different parameter configurations of the same circuit during the circuit exploration and only validate the good output candidates via Spice simulations.

Note that all these mechanisms to reduce circuit evaluation costs are applicable not only to our UCT-based framework, but also work for other circuit generation approaches. In fact, we use the de-isomorphism method with special circuit encoding and hash table and the hybrid circuit evaluation strategy for all the evaluated circuit generation approaches when conducting the experimental comparison in Section 7.

6.1 Reducing Isomorphic Circuits via Unique Encoding and Hash Table

As discussed in Section 3, a converter topology is essentially a graph. Randomly generated topologies will lead to numerous isomorphic graphs that hinder the effectiveness of finding good topologies. Even with action pruning techniques applied to our UCT-based framework for reducing invalid and isomorphic topologies, the circuits generated by UCT still contain many isomorphic ones. Since circuit evaluation is very time-consuming, we design an automatic de-isomorphism method to further remove redundant circuit evaluation for isomorphic circuits.

Unique Encoding. The first step of our de-isomorphism method is to assign a unique encoding, also called *key*, to each circuit. While the topology representation describes a circuit abstractly, the actual implementation of a circuit generation algorithm can arbitrarily number the chosen devices and index their interconnection ports. Depending on how the devices are selected and ports are connected, the same circuit can end up with different concrete topology representations.

One major source of isomorphism is the component indices. During topology generation, we will assign indices to every component, which are then inherited by the chosen devices. As a result, the same circuit may have different topology representations with different device indices. Our action pruning techniques in Section 4.3 reduce such isomorphic topologies for the UCT-based framework, but other circuit generation approaches still suffer from this issue severely. To address

this problem, we use a string-type encoding where the components are identified by the name of the chosen device type, e.g., L and C. For circuits with multiple components with the same device type, these components are identified by the name of the device type followed by an index, e.g., L1 and L2.

Fig. 7. The same (partial) circuit with two different topology representations with different edges.

In addition to isomorphism caused by components, another source of isomorphism comes from edges connecting ports. For instance, Figure 7 shows two different (partial) topology representations with different edges connecting different ports. However, they actually represent the same circuit, since the two edges in the respective graph connect the same three ports together. To reduce the isomorphism introduced by the indexed ports, our algorithm will first combine all ports that are connected together as one port by assigning it with the same identifier. Specifically, it replaces the port index with a string-type encoding, which is constructed by concatenating the string-type identifiers of the connected components. The concatenated identifiers are also ordered based on the device types and additional indices. For example, Ports 0, 2, and 3 in Figure 7 will be assigned with the same identifier "L-C-Gnd" indicating they are the same combined port.

Then the *key* is a string-type list of the circuit topology networks in the alphabet order. Each network is an encoded component with its string-type encoded left and right port. The order of the two ports is also in alphabet order. In this way, the *key* can uniquely represent a circuit by efficiently encoding the topology information into a string.

Hash Table. The second step of our de-isomorphism method uses a hash table to keep track of all unique circuits that have already been evaluated. Note that both the UCT-based framework and baseline algorithms may query the circuit evaluation about a topology more than once. To avoid unnecessary circuit evaluation costs, we implemented a hash table to cache the efficiencies and conversion ratios of the topologies that have been queried during each experiment. Therefore, in our experiments for both the proposed algorithm and baselines, all the queries to the circuit evaluation are only about unique topologies. The use of hash table is especially beneficial for the baselines like Genetic Search, as they cannot make use of action pruning to avoid some isomorphic topologies by construction.

There is a trade-off between time and space when using the hash table. Specifically, it takes $0.43\mu s$ on average to search the hash table for a specific topology and duty cycle. As a comparison, the State-Space Averaging method and high-fidelity simulation consume 1.95 seconds and 12.45 seconds on average to evaluate a circuit, respectively. On the other hand, it takes additional memory space to store the hash table. For example, a hash table for 5-component circuits with 1.55×10^5 entries may consume up to 323.1MB to store. Note that, although the hash table size increases for longer exploration and higher circuit complexity, the circuit evaluation cost also increases enormously. Therefore, using a hash table during the UCT search significantly reduces overall time cost with an affordable trade-off in memory.

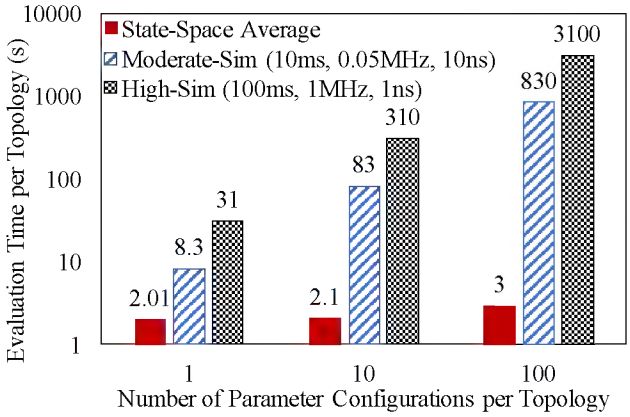


Fig. 8. Evaluation time per topology for different numbers of parameter configurations per topology under State-Space Averaging method, moderate-fidelity simulation (with 10ms transient time, 0.05MHz switching frequency, and 10ns minimum time-step), and high-fidelity simulation (with 100ms transient time, 1MHz switching frequency, and 1ns minimum time-step).

6.2 Efficient Evaluation via State-Space Averaging

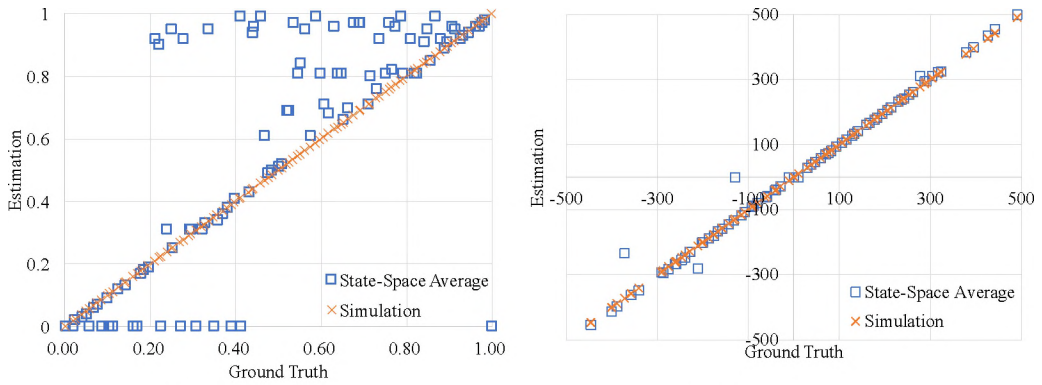
The reward function used by UCT is a computational bottleneck as it requires evaluating different control parameters for different topologies sampled by all the rollouts. Simulation software, such as NGSpice [28], often takes a long time to evaluate one topology with fixed parameters. To speed up the circuit evaluation, we instead adopt the *State-Space Averaging* approach [30]. This approach shares the major computation among the circuits with the same topology but different parameters, which makes it a faster surrogate model in estimating a large number of parameters.

Specifically, State-Space Averaging characterizes the transfer properties of switching stages of a power converter circuit. By dividing the circuit into phases-I and phase-II sub-circuits, deriving state-space equations, and averaging state changes with the corresponding duty-cycles (i.e., d and $1 - d$), it derives the estimated output voltage and power efficiency.

We compared the computation times of State-Space Averaging versus NGSpice. Figure 8 shows that the evaluation time per topology under State-Space Averaging is significantly lower compared to moderate-fidelity and high-fidelity simulations. The advantage of State-Space Averaging is even higher when more parameter configurations of one topology (e.g., the same topology with different duty cycles) need to be evaluated. This is because deriving state-space equations is the most expensive step of State-Space Averaging, but these equations remain the same as long as the topology remains the same. Hence, increasing the number of parameter configurations per topology does not increase the evaluation time much under State-Space Averaging, while simulations must be performed for each parameter configuration of the same topology.

Despite its speed advantage, State-Space Averaging has a disadvantage — it is only accurate if the circuit is linear during operations. Nonlinear effects, such as discontinuity in conduction or enforced switch dead-zone, will introduce errors. As such, the state-space method usually overestimates the efficiencies of topologies.

We conduct an in-depth analysis to examine the accuracy of the State-Space Averaging method compared to the ground-truth high-fidelity simulation on various 5-component circuits. Figure 9 reveals that the State-Space Averaging method can consistently estimate the output voltages of circuits, although most estimations are slightly higher. However, we can notice that for circuits



(a) Power Efficiency (b) Output Voltage

Fig. 9. Performance evaluation under the State-Space Averaging method (blue diamonds) compared to the ground truth high-fidelity simulation (orange crosses). As the evaluation results of the high-fidelity simulation are the ground truth, its estimated values are the same as the ground truth. In contrast, the estimated values given by the State-Space Averaging method can deviate from the ground truth values.

with high power efficiencies, the State-Space Averaging method often overestimates the efficiencies. For some circuits with low efficiencies, it also cannot correctly estimate the efficiencies.

(a) Buck-boost (conversion ratio -2) (b) Buck (conversion ratio 0.5) (c) Boost (conversion ratio 2)

Fig. 10. Reward estimation under the State-Space Averaging method (blue diamonds) compared to the ground truth high-fidelity simulation (orange crosses) for buck-boost, buck, and boost converters, respectively.

We also look into how the error in power efficiency estimated by the State-Space Averaging method affects the reward estimation for different types of converters. Using NG-spice simulation result as ground-truth, we measured predicted reward for buck-boost, buck and boost converters using State-Space Averaging approach. From the result and Figure 10, we notice that the State-Space Averaging method can more accurately estimate the rewards for boost converters. Its estimations for buck converter deviate most from the ground truth values. This suggests that using the State-Space Averaging method to replace the high-fidelity simulation may not work well. The evaluation time per topology is illustrated in Figures 8

To address this issue, we take a hybrid circuit evaluation approach. During topology search, a topology is evaluated by State-Space Averaging. With the optimistic nature of State-Space Averaging, it covers almost all qualified topologies with high rewards. Once the top topology candidates are generated, NGSpice simulation will be invoked to offer ground-truth evaluation for the final topology and control scheme selection.

7 EVALUATION

We evaluate our framework across a spectrum of custom design tasks with different conversion ratios and compare its performance with baseline algorithms. We also conduct an ablative study on the effectiveness of the proposed data-driven default policy. Next, we evaluate the effectiveness of our parallelization techniques and the hybrid circuit evaluation approach. Finally, we discuss the nonconventional power converter topologies discovered by our framework.

7.1 Experiment Setup

We conduct our evaluation on 3 servers with same configurations. Each server consists of Ubuntu 18.04, 196GB RAM, and two Intel(R) Xeon(R) Gold 5120 CPU, each CPU has 14 cores. We use Python 3.8 to implement our topology generation program. We conduct the evaluation on the power converter design task with five components, each with two ports. Together with the three external ports (`Vin`, `Vout`, and `Gnd`), the design space contains topologies with a total of thirteen ports. The component device types have fixed device parameters and include capacitors `C` ($10\mu F$), inductors `L` ($100\mu H$), phase-I switches S_a, and phase-II switches S_b For external ports, we consider an input resistor of 0.1Ω for `Vin` and an output resistor of 100Ω and an output capacitor of $10\mu F$ for `Vout`. The candidate control parameter (i.e., duty cycle) ranges from 0.1 to 0.9, with a step size of 0.1. We configure the frequency as $1MHz$ for both analytic evaluation and simulation. We set the input voltage to $100V$, and the target voltage outputs are chosen from the range of -$300V$ to $300V$. Additionally, a topology with a conversion ratio smaller than -5 or larger than 5 will be considered invalid. In NGSpice simulation, the transient simulation time is set as $60s$.

7.2 Implementation Details and Competitive Algorithms

In this section, we describe the implementation details of our UCT-based framework and the baseline algorithms. To the best of our knowledge, *genetic search* and *exhaustive search* are the only existing machine learning algorithms that might not need non-trivial domain knowledge and are applicable to the converter circuit topology design, as discussed in Section 2. Hence, we implemented *genetic search* and *random search* (i.e., the randomized version of *exhaustive search*) as the baseline algorithms for comparison.

Implementation Details of Proposed Algorithm. There are two important implementation details of our UCT-based topology generation algorithm. First, we use the number of rollouts to decide the number of explorations each run of the UCT-based topology generation. In each rollout, we expand a UCT tree-node and execute a rollout to get a complete topology. However, the number of rollouts can be larger than the number of queries, since we only query the circuit simulation when a rollout leads to a valid topology that is not in the hash table, as described in Section 6.1. Second, different from the common UCT design that completely reconstructs a new tree for each step, we inherit the sub-tree corresponding to the selected action from the tree of the previous step to make full use of the collected information. This operation makes our UCT-DP algorithm converge faster.

Random Search (RS). Random Search is a strategy that starts with an empty topology, randomly selects device types, and then randomly connects ports until reaching a complete topology. Note

that, if a port already has connections to other ports, then it is not required to connect to yet another port. Thus, in this case, RS may skip adding more connections to this port with a probability of 0.8. RS uses the same reward function as UCT, along with the same prohibitive paths that prevent generating invalid topologies. After searching a prefixed number of complete circuit topologies, RS outputs the one that has the highest reward.

Genetic Search (GS). We implemented another popular heuristic-based search algorithm, namely Genetic Search, for the power converter design task [26]. GS starts with 15 random topologies. In each round, GS selects 3 topologies with the highest rewards from the current generation and uses them as parents to generate offspring by mutation. We implemented the following mutation types: (1) A crossover randomly selects a component that exists in two parents and exchanges the connections of this component [31]; (2) A component change randomly changes the device type of a random component in a parent; (3) A connection insertion randomly selects and connects two unconnected ports in a parent; (4) A connection removal removes a random connection of a parent; (5) A connection switch selects a random connection of a parent and changes one of the connection's ports to another random port; In each round, the above mutation types are chosen with probabilities of {20%, 10%, 30%, 20%, 20%}, respectively.

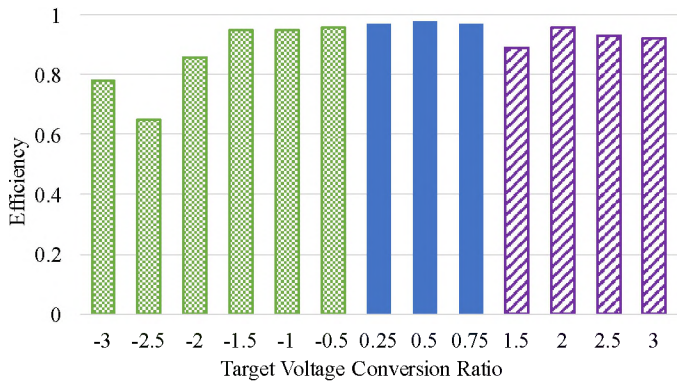


Fig. 11. Average power efficiencies of power converters generated by the proposed framework for custom design tasks with increasing voltage conversion ratios. From left to right in different colors and shades, the target converter types are buck-boost, buck, and boost.

7.3 Evaluating UCT-based Circuit Generation

In this Subsection, we conduct experiments to evaluate our UCT-based circuit generation and compare it against the above baseline methods. To form a focused evaluation and fair comparison with baselines, all the experiments in this Subsection use the same de-isomorphism method (Section 6.1) and hybrid circuit evaluation approach (Section 6.2) for (sequential) UCT and baselines.

Custom Design Tasks. We first evaluated our UCT-based framework on different custom design tasks for power converters. The design tasks can be classified into three settings, including buck-boost, buck, and boost converters. For each target voltage conversion ratio, we run our framework five times and compute the average efficiency of the generated topologies. In Figure 11 shows the average power efficiencies under our framework for different voltage conversion ratios. For the Buck-Boost, Buck, and Boost converter, we set the target conversion ratio as -2, 0.5 and 2 respectively. Both the baselines and the proposed method use the hash table to decrease the number of queries. The UCT-based topology generation uses the State-Space Averaging approach to get

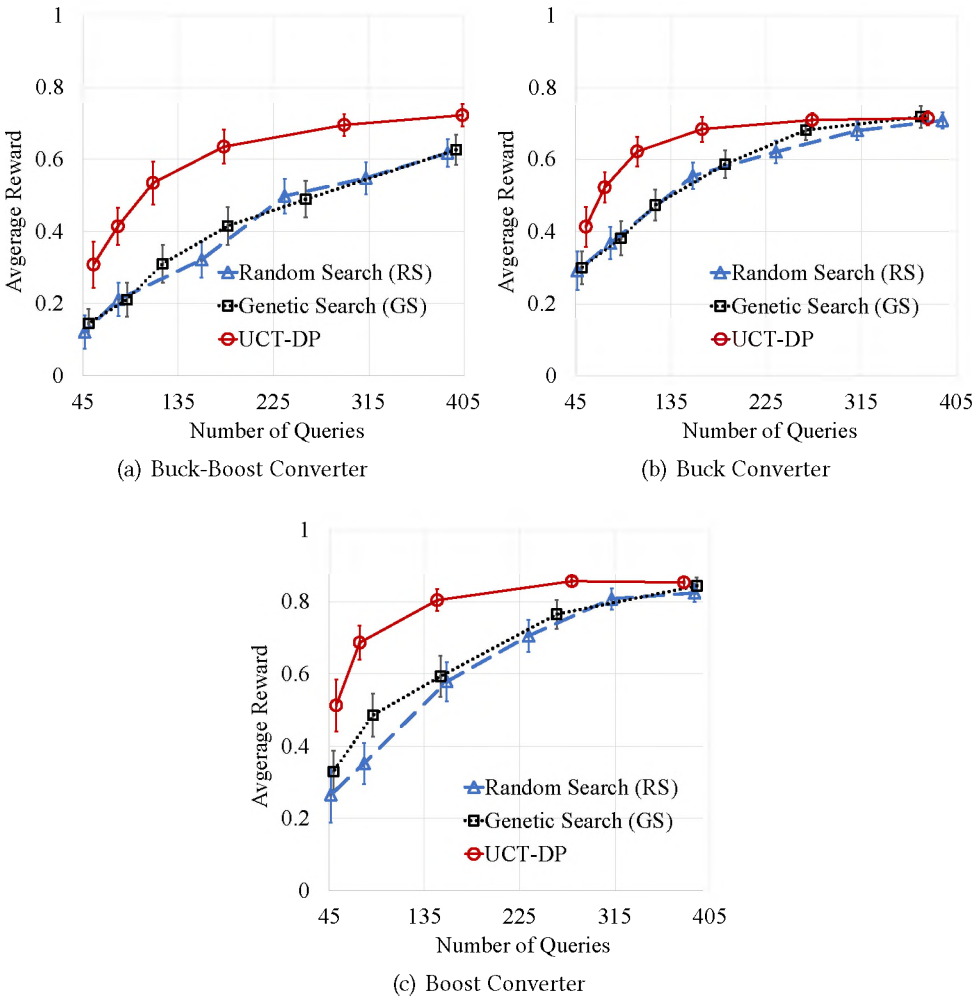


Fig. 12. Average rewards with 95% confidence interval under UCT-DP vs. GS vs. RS with increasing numbers of unique queries to circuit evaluations. The x-axis shows increasing numbers of queries to circuit evaluations for unique topologies. The y-axis is the obtained reward calculated using the power efficiency and voltage conversion ratio of the best candidate circuit generated by an algorithm, averaging from 200 runs.

the utility the circuits and conduct simulation on the top 5 topologies that has the highest utilities, while the genetic and random search approach directly use the high-fidelity simulator to accurately evaluate the quality of the explored circuit topologies. For each circuit topology, we sweep the candidate duty cycles mentioned in the experiment setting and uses the maximum utility as the utility of this circuit topology. Note that the horizontal axis is the number of topology queries. During conducting the experiments, we run each setting 200 times in the sequential way and collect the average rewards and number of queries (newly explored topologies) to show the exploration efficiency of different methods.

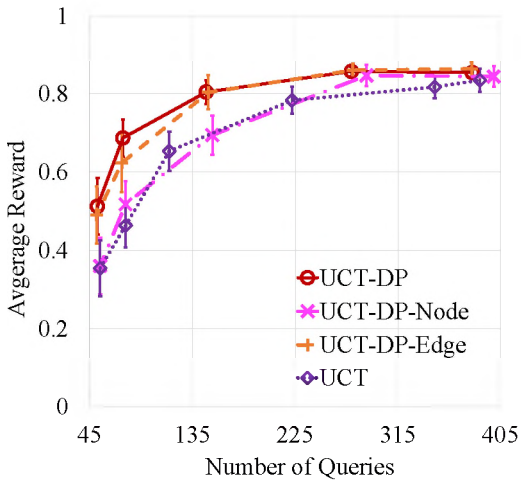


Fig. 13. Ablative studies of UCT-based approaches without default policy (UCT), with node selection default policy (UCT-DP-Node), with edge selection default policy (UCT-DP-Edge), and with both node and edge selection default policies (UCT-DP) for boost converters.

Results show that our framework can successfully find high-performing topologies for most settings. The average efficiencies for smaller conversion ratios tend to be lower, mainly because these buck-boost converters are more sensitive to their duty cycles. For example, if we change the step size of the duty cycle from 0.1 to 0.05, our framework can discover a better topology with an efficiency of 0.89 for the design task with a conversion ratio of -2.5.

Comparison with Baseline Algorithms. We compared our UCT-based approach with RS and GS on buck-boost, buck, and boost converters. Since all these algorithms are anytime algorithms (i.e., the performance monotonically increases as more computation is used), we compared their performances conditioned on their computation costs, measured by the number of queries to circuit evaluations. Figure 12 reports the average reward of topologies generated by each algorithm. For all types of converters, our UCT-based approach, named **UCT-DP**, outperforms both RS and GS, while GS is comparable with or slightly outperforms RS. When the number of queries is around 110, UCT-DP achieves 83%, 35%, and 37% higher rewards compared with GS for buck-boost, buck, and boost converters, respectively. In terms of computation efficiency, results show that, compared to GS, UCT-DP needs up to 63%, 52%, and 67% fewer queries to obtain the same average rewards for buck-boost, buck, and boost converters, respectively. We also observe that all the algorithms (especially RS and GS) perform slightly worse for buck-boost converters, which may be caused by the step size of duty cycles as discussed above. Overall, the results demonstrate the efficacy of our UCT-based approach to discover high-quality topologies using only a small number of queries.

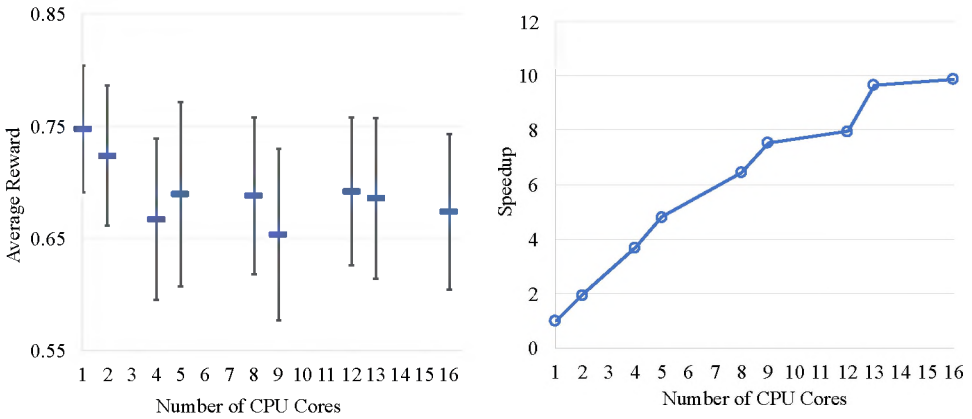
Ablative Studies on Default Policies. In Section 4, we described how we incorporate offline knowledge using default policies. In this experiment, we examine the effectiveness of the default policies by performing an ablation study. Figure 13 presents the average rewards with and without the two default policies, namely node selection and edge selection, for the boost converter design task. The results for other power converter types are similar. Not surprisingly, UCT-DP with both default policies performs the best. We also observe that UCT with edge selection only (UCT-DP-Edge) performs significantly better than with node selection only (UCT-DP-Node). This is partly because choosing good connections among the exponentially many connections is more challenging.

Moreover, since device type selections are performed in the first phase before connection selections, the node selection default policy is only used by the rollouts during the first phase. In contrast, the edge selection default policy is used by all the rollouts, so it has a higher impact on the performance.

7.4 Evaluating Parallel Circuit Generation

We evaluate the performance of the proposed two parallelization schemes, root parallelization and leaf parallelization, on circuit generation using different numbers of CPU cores. Due to the GIL (Global Interpreter Lock) in python 3.8, our implementation uses the multiprocessing package to generate multiple processes (instead of multiple threads) to implement the parallel UCT-based framework. To efficiently share the information between the processes, we use the manager in multiprocessing to maintain the rollout counter and the shared reward hash table. According to our measurements, the cost of information synchronization using the manager in multiprocessing is relatively low, so it has little impact on the performance of the parallelization schemes.

In the following experiments, the circuit design task is almost the same as the settings in the previous subsection. The only difference is that here we only use 5 candidate duty cycles: 0.1, 0.3, 0.5, 0.7, and 0.9. We still use the hybrid circuit evaluation approach described in Section 6.2 with parallelization of the final high-fidelity simulation on the top 5 circuit topologies to make full use of the computation resource. For root parallelization, the parallel look-ahead trees share the reward hash table and the rollout counter, as described in Section 5.3. For each setting, we conduct 60 runs and collect the average reward and average running time. The speedup is then calculated by dividing the average running time of parallel execution by that of sequential execution. In the experiments, we vary the number of cores used in the execution, ranging from 1 up to 16.



(a) Average Reward with 95% confidence interval (b) Speedup

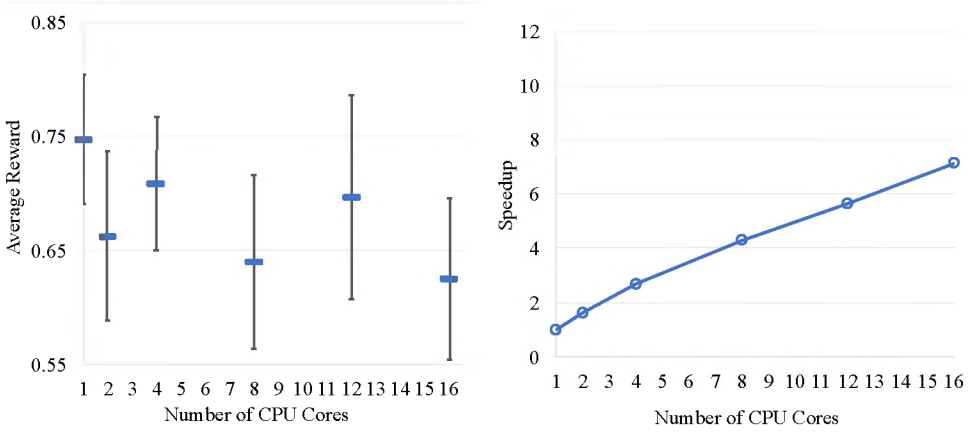
Fig. 14. Performance of root parallelization scheme on varying numbers of cores

Performance of Root Parallelization Scheme. Figures 14(a) and 14(b) show the average reward and speedup of UCT-based circuit generation using root parallelization, respectively. Results show that root parallelization achieves relatively large speedup (e.g., a speedup of 9.7 on 13 cores) by making efficient use of the multiple cores with little performance loss in terms of the average efficiency of the generated topologies.

In fact, the speedup of root parallelization approaches increases, if the final high-fidelity simulation time is not included in the total running time. This is because the number of queries to the

high-fidelity simulation is equal to the 5 top topologies, each with 5 duty cycles, which is equal to 25. Additionally, the simulation times for different circuits may also vary. Thus, it is relatively hard to evenly distribute the computation among multiple cores, especially when the number of cores is large. The speedups on 8, 9, 12, and 13 cores confirm this observation. On 8 cores, it is likely to have one core with one more query to perform than all the other cores, while this additional query can be executed by an additional core in parallel on 9 cores. Hence, the speedup jumps slightly from increasing the number of cores from 8 to 9. What's more, we compare the speedups of root parallelization using 16 CPU cores with a total of 90 rollouts and 1440 rollouts. The speedup of a large number of rollouts reaches 10.64 while the speedup of the small number of rollouts is only 8.78. This is because when the number of rollouts is small and the number of cores is large, the total topology exploration time decreases dramatically and the negative impact of parallelization overhead becomes more obvious.

While parallelization schemes significantly reduce the total execution time, they also come with a cost of performance loss. For root parallelization, this is due to the fact that multiple trees can only communicate a limited amount of information between steps and do not share all the global information, in order to reduce the synchronization overheads. From the results of average rewards, we can see that the performance of root parallelization drops slightly with more cores used. Additional experiments reveal that root parallelization can achieve comparable performance with sequential UCT if performing 20% more queries. Since these queries use the fast State-Space Averaging method under our hybrid circuit evaluation approach, the total running time only increases by about 6%, which is almost negligible considering the large speedup.

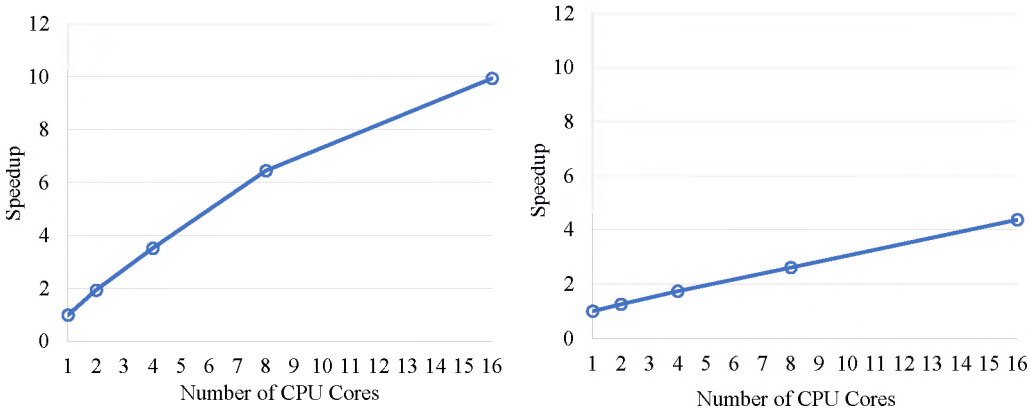


(a) Average Reward with 95% confidence interval (b) Speedup

Fig. 15. Performance of leaf parallelization scheme on varying numbers of cores

Performance of Leaf Parallelization Scheme. We can see that both the average reward and speedup of leaf parallelization are smaller compared to root parallelization. For instance, using the same 16 cores, leaf parallelization only achieves 7.1x speedup with around 16.6% performance loss. In terms of speedup, leaf parallelization needs to synchronize for every tree-node expansion, while root parallelization only synchronizes once every step with multiple node expansions. Hence, leaf parallelization requires finer-grained load balancing, which is harder compared to root parallelization.

The performance drop, in terms of average reward, of leaf parallelization is due to reasons different from root parallelization. While leaf parallelization maintains a single look-ahead tree containing all the global information, in order to run multiple rollouts in parallel on all available cores, these rollouts are performed to estimate the score of the same state. However, for some states that have very low scores and never produce good circuits, it may not be necessary to perform these many rollouts. Instead, spending some of these rollouts on other states may be more beneficial to finding better circuits.



(a) Root Parallelization (b) Leaf Parallelization

Fig. 16. Speedup of root and leaf parallelization schemes for our UCT-based framework using only the high-fidelity simulation as circuit evaluation method.

Performance of Root and Leaf Parallelization Schemes using Simulation. Figure 16 shows the speedup of root and leaf parallelization schemes for our UCT-based framework using only the high-fidelity simulation. We still conduct this evaluation on the 5-component converter circuit topology generation task. The only difference is that we do not sweep duty cycles but we add one additional phase with actions for choosing the duty cycle, just as what we discussed in Section 4.1. We can see that the speedup achieves by root parallelization with the simulator is comparable to the hybrid evaluation, while the speedup under leaf parallelization decreases significantly. This is mainly because the larger variation of simulation times for different circuit exaggerate the blocking time due to parallel rollouts. This significantly reduces the speedup achieved by leaf parallelization when using only simulation.

In addition, we also evaluate the speedups of the scheme that using high-fidelity simulator and sweeping all the duty cycles for each topology. It shows that when using 8 cores, the root and leaf parallelization only achieve around 4x and 2.4x speedup respectively, which are lower than the scheme without duty cycle sweeping. This is mainly because sweeping the duty cycles makes the load-balance between the cores harder to achieve, as both the total length and variation of the evaluation time for one query become larger.

Ablative studies on Root Parallelization. To investigate the impact of different additional parallelization techniques applied to root parallelization, we conduct an ablative study where the number of CPU cores is fixed to 8 with the hybrid circuit evaluation approach. We respectively remove the following techniques from root parallelization and examine the performance: (1)

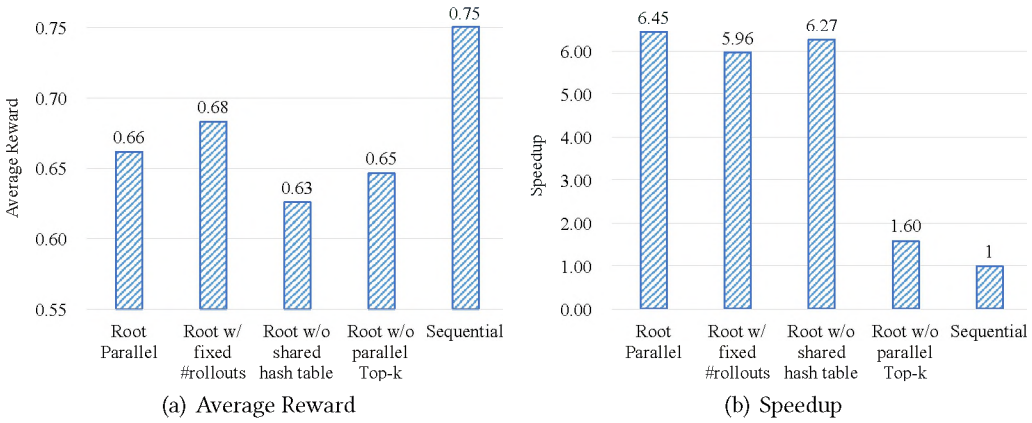


Fig. 17. Performance of root parallelization without different additional parallelization techniques

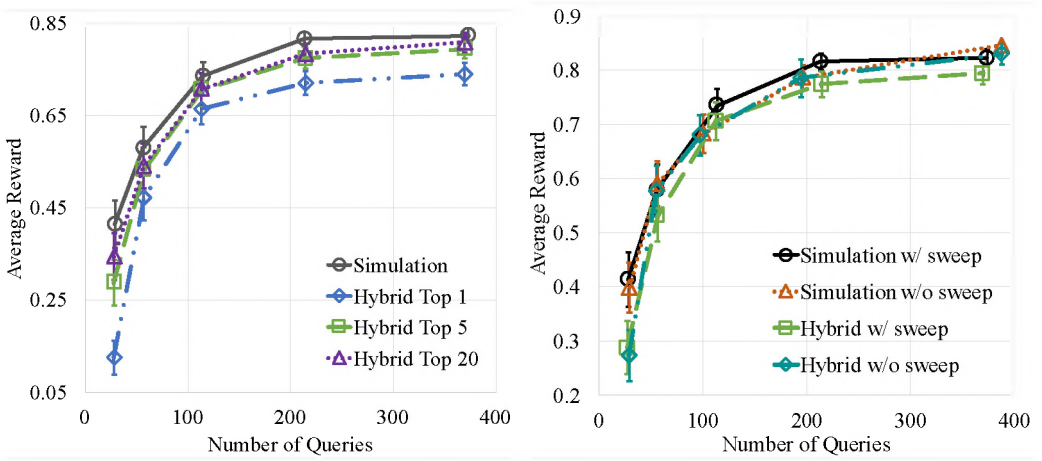
removing the rollout counter and assigning the same fixed number of rollouts to each look-ahead tree in root parallelization; (2) disabling the use of reward hash table shared by multiple trees, but letting each tree maintain its own hash table; (3) running the top-k high-fidelity simulation sequentially without parallelism. For each setting, we conduct 30 runs and collect the average reward and the speedup.

Figure 17 presents the result of the ablative study. We can observe that fixing the number of rollouts and evenly splitting them for each tree reduces the speedup slightly while increasing the average reward slightly. This is simply because the computation load balancing is worse while all trees are more evenly expanded, as discussed in Section 5.3. This impact is more evident when using only the high-fidelity simulation for all circuit evaluations, since the execution time variation is higher for simulation than the State-Space Averaging method. While the speedup barely changes without using the shared hash table between trees, the average reward drops the most. This confirms that sharing more global information is helpful.

The most significant impact comes from running the top 5 topologies sequentially or in parallel. Parallelizing the circuit simulating is essential for achieving good speedup for both root and leaf parallelization. As shown in Figure 17(b), if simulating the top 5 topologies sequentially, the speedup using 8 cores is only 1.6. This is because high-fidelity simulation is much more time-consuming compared with the State-Space Averaging method. According to our measurement, in the sequential UCT-based circuit generation, the high-fidelity simulation cost around 72.2% of the total running time. Thus, sufficiently paralleling the high-fidelity simulation is critical for achieving good speedup.

Table 1. Running Time (second) of different circuit evaluation approaches

#Query / Approach	138	284	566	1072	1849
Simulation	2160.75	4242.37	8576.25	16034.62	28001.62
Hybrid Top 1	66.43	123.29	233.01	430.51	733.56
Hybrid Top 5	116.23	173.09	282.81	480.31	783.36
Hybrid Top 20	302.98	359.84	469.56	667.06	970.11



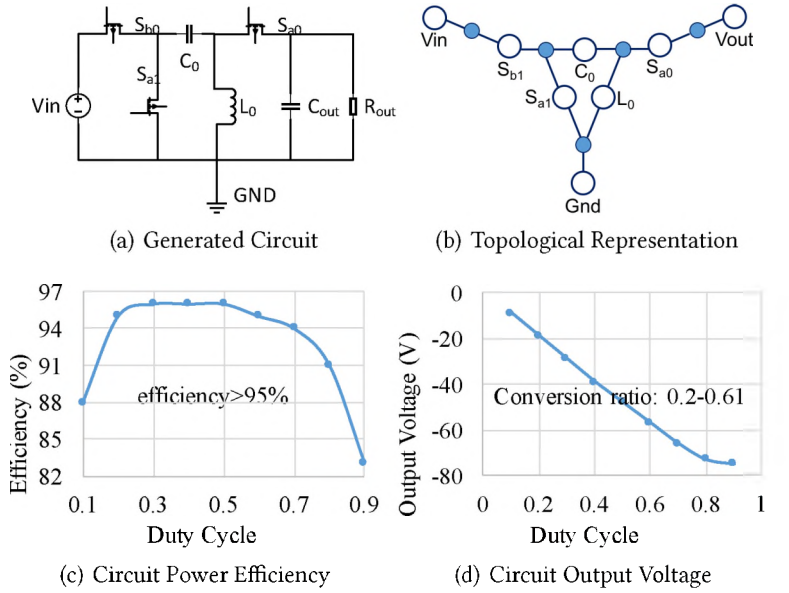
(a) Simulation vs. Hybrid with Different Top k (b) With and Without Sweeping Duty Cycles

Fig. 18. Performance of different circuit evaluation approaches

7.5 Evaluating the Effectiveness of Hybrid Circuit Evaluation Approach.

As the State-Space Averaging method cannot very accurately evaluate circuits, we conduct the following experiments to validate our hybrid circuit evaluation approach in Figure 18(a). For the same experiment setting, we compare the performance between our UCT-based framework using only the high-fidelity simulation for all circuit evaluations and the same framework using the hybrid circuit evaluation approach. Additionally, for the hybrid approach, we also vary the number of top topologies validated by the high-fidelity simulation. Note that Hybrid Top 1 is essentially using only the State-Space Averaging method for circuit evaluation. Specifically, under Hybrid Top 1, UCT generates the circuit that is considered the best-performing one based on State-Space Averaging evaluation. The ground-truth performance of this circuit is then evaluated by the high-fidelity simulation and used to calculate the average reward in Figure 18(a). Not surprisingly, the average rewards under the hybrid approach are slightly lower than those under high-fidelity simulation. Moreover, by increasing the number of topologies deemed high-quality by the State-Space Averaging method during the circuit exploration and validated by the high-fidelity simulation, this performance gap decreases. Validating the top 5 topologies is relatively sufficient to achieve almost the best performance under the hybrid approach.

Table 1 shows the average running times of different circuit evaluation methods when the number of queries performed by UCT increases. We can observe that the running time of Simulation and Hybrid Top 1 (i.e., UCT with State-Space Averaging evaluation only) increases nearly linearly with the increase of the number of queries, since the time to perform the circuit evaluation dominates the total running time. As the State-Space Averaging method is much faster than the simulation, the running time of Hybrid Top 1 (i.e., UCT with State-Space Averaging evaluation only) has a 33x to 38x speedup compared to Simulation at the cost of performance degradation. In contrast, Hybrid Top 5 and Top 20 achieve a better performance-cost trade-off. By performing some additional high-fidelity simulations, Hybrid Top 5 and Top 20 significantly improve the performance of generated circuits, while the total running times only have a constant increase and are still significantly smaller than Simulation. For example, the total running times of Hybrid Top 5 and Top 20 are more than 28 times faster than that of Simulation with little performance loss. In summary, the



(a) Generated Circuit (b) Topological Representation

(c) Circuit Power Efficiency (d) Circuit Output Voltage

Fig. 19. An example of unconventional power converter circuit discovered by our framework.

hybrid evaluation methods can achieve the same level of performance compared with using the high-fidelity simulation with significantly reduced computation cost.

Section 4.1 discusses the formulation choice regarding whether to sweep all control parameters to find the one with maximum utility. Figure 18(b) presents the comparison between the hybrid approach and high-fidelity simulation with and without sweeping duty cycles under the same setting. Note that for the cases with sweeping duty cycles, a query is defined as evaluating one topology with all 5 duty cycles. In contrast, for cases without sweeping duty cycles, the circuit evaluation is queried for each topology with a specific duty cycle. To have a fair comparison, we divide the number of queries for the latter case by 5, so that the number approximates that of the topology queries. Results show that the performance under formulation with and without sweeping the control parameter is comparable. Therefore, it confirms that finding good topologies is more critical than tuning parameters for circuit generation tasks.

7.6 Discussion on New Topologies

Our automatic power converter design framework generates topologies that meet design targets. Besides classical power converter topologies, it is also able to find interesting unconventional topologies that both satisfy specifications and have high power efficiencies, such as the one in Figure 19. These automatically generated topologies have the potential to shed light on fundamental circuit innovations. With close collaboration with human experts, our framework can help to discover innovative circuits that have not been studied.

7.7 Discussions on the Framework Constraints and Potentials

The proposed framework, despite the advanced performance, has its limitations and unexplored potentials. First, the proposed framework is an early attempt at automated topology design that targets structure exploration. Thus, after the topologies are generated, fine-grained parametric

analyses, such as comprehensive parameter sweeping, LC-coupling, and PVT variation, are needed to validate their performance (e.g., efficiency and robustness) in real-world applications.

Second, our framework currently targets only power converter circuits, but it has the potential to extend to other similar circuits with minimal algorithmic adjustment. For example, switch-capacitor-based digital-to-analog/analog-to-digital data converters share similar component pools (e.g., switches across phases, capacitors with various parameters) and input/output (e.g., linearity) analytical approach. It is of great interest to further explore the versatility of the proposed framework on general analog/mixed-signal circuit topology design problems.

Third, although theoretically there is no limitation on the maximally-achievable circuit complexity, the time cost of generating high-quality circuits is highly correlated to the circuit complexity for the following reasons. The circuit topology space increases combinatorially. For example, from our empirical measurement, we observe that the number of unique circuits expands from 2×10^5 to $\times10^7$ when the number of components increases from 5 to 6 components. Additionally, the circuit evaluation cost increases significantly when the circuit has a much larger number of devices. For instance, the high fidelity simulation using NGSpice takes about 4.29 seconds on average to simulate a 3-component converter, while it needs 12.45 seconds for a 5-component converter. Moreover, the ratio of well-performing circuit candidates reduces from 2.29% to 0.36% when the number of components increases from 3 to 5 components. Note that these difficulties exist for any circuit topology generation approaches. Therefore, additional techniques and advancements, such as an accurate neural network model for fast evaluation of arbitrary circuit topologies, are needed to scale up to much larger circuits.

8 CONCLUSION

In this work, we proposed a parallel UCT-based power converter topology generation framework, which explores the design space automatically. We incorporated physics-informed constraints and data-driven default policies to reduce the design space and improve the efficiency of our framework. We implement several parallelism techniques to speed up the execution time of circuit exploration. Additionally, we adopted a hybrid circuit evaluation using both the fast State-Space Averaging method and the accurate high-fidelity simulation. Finally, evaluations showed that our framework can generate near-optimal circuit topology for buck-boost, buck, and boost converters. Compared to the alternative approaches, our framework can discover better circuit topologies with reduced computational costs. As future work, we plan to apply our framework to other analog/mixed-signal circuit topology design problems and explore using deep-learning models to speed up the circuit evaluation time further.

ACKNOWLEDGMENTS

The information, data, or work presented herein was funded in part by the Advanced Research Projects Agency-Energy (ARPA-E), U.S. Department of Energy, under Award Number DE–AR0001210 and in part by the U.S. National Science Foundation, under Grant Number CNS–1948457. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

REFERENCES

[1] Zhaori Bi, Dian Zhou, Sheng-Guo Wang, and Xuan Zeng. 2017. Optimization and quality estimation of circuit design via random region covering method. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 23, 1 (2017), 1–25.

[2] Amine Bourki, Guillaume Chaslot, Matthieu Coulm, Vincent Danjean, Hassen Doghmen, Jean-Baptiste Hoock, Thomas Hérault, Arpad Rimmel, Fabien Teytaud, Olivier Teytaud, et al. 2010. Scalability and parallelization of monte-carlo tree search. In *International Conference on Computers and Games*. Springer, 48–58.

[3] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games* 4, 1 (2012), 1–43.
[4] Tristan Cazenave and Nicolas Jouandeau. 2007. On the parallelization of UCT. In *Computer games workshop*.
[5] E. Chang, J. Han, W. Bae, Z. Wang, N. Narevsky, B. NikoliC, and E. Alon. 2018. BAG2: A process-portable framework for generator-based AMS circuit design. In *2018 IEEE Custom Integrated Circuits Conference (CICC)*. 1–8.
[6] Guillaume MJ-B Chaslot, Mark HM Winands, and HJVD Herik. 2008. Parallel monte-carlo tree search. In *International Conference on Computers and Games*. Springer, 60–71.
[7] H. Chen, K. Zhu, M. Liu, X. Tang, N. Sun, and D. Z. Pan. 2020. Toward Silicon-Proven Detailed Routing for Analog and Mixed-Signal Circuits. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–8.
[8] Rémi Coulom. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games*. Springer, 72–83.
[9] Shaoze Fan, Ningyuan Cao, Shun Zhang, Jing Li, Xiaoxiao Guo, and Xin Zhang. 2021. From Specification to Topology: Automatic Power Converter Design via Reinforcement Learning. In *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–9.
[10] Sylvain Gelly and David Silver. 2007. Combining online and offline knowledge in UCT. In *Proceedings of the 24th international conference on Machine learning*. 273–280.
[11] Helmut Graeb, Stephan Zizala, Josef Eckmueller, and Kurt Antreich. 2001. The sizing rules method for analog integrated circuit design. In *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No. 01CH37281)*. IEEE, 343–349.
[12] Tobias Graf, Ulf Lorenz, Marco Platzner, and Lars Schaefers. 2011. Parallel Monte-Carlo tree search for HPC systems. In *European Conference on Parallel Processing*. Springer, 365–376.
[13] Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. 2014. Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. *Advances in neural information processing systems* 27 (2014).
[14] Kourosh Hakhamaneshi, Nick Werblun, Pieter Abbeel, and Vladimir Stojanović. 2019. Bagnet: Berkeley analog generator with layout optimizer boosted with deep neural networks. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.
[15] K. Hakhamaneshi, N. Werblun, P. Abbeel, and V. Stojanović. 2019. Late Breaking Results: Analog Circuit Generator based on Deep Neural Network enhanced Combinatorial optimization. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*. 1–2.
[16] S. Han, S. Jeong, C. Kim, H.-J. Park, and B. Kim. 2020. GUI-Enhanced Layout Generation of FFE SST TXs for Fast High-Speed Serial Link Design. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 1–6.
[17] Jintae Kim, Jaeseo Lee, Lieven Vandenberghe, and Chih-Kong Ken Yang. 2004. Techniques for improving the accuracy of geometric-programming based analog circuit design optimization. In *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.* IEEE, 863–870.
[18] Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*. Springer, 282–293.
[19] Karl Kurzer, Christoph Hörtnagl, and J Marius Zöllner. 2020. Parallelization of monte carlo tree search in continuous domains. *arXiv preprint arXiv:2003.13741* (2020).
[20] Wook Lee and Frans A. Oliehoek. 2020. Analog Circuit Design with Dyna-Style Reinforcement Learning. In *NeurIPS 2020 Workshop: Machine Learning for Engineering Modeling, Simulation, and Design*.
[21] Anji Liu, Jianshu Chen, Mingze Yu, Yu Zhai, Xuewen Zhou, and Ji Liu. 2019. Watch the Unobserved: A Simple Approach to Parallelizing Monte Carlo Tree Search. (2019).
[22] M. Liu, W. Li, K. Zhu, B. Xu, Y. Lin, L. Shen, X. Tang, N. Sun, and D. Z. Pan. 2020. S3DET: Detecting System Symmetry Constraints for Analog Circuits with Graph Similarity. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*.
[23] Yi-Chen Lu, Sai Surya Kiran Pentapati, Lingjun Zhu, Kambiz Samadi, and Sung Kyu Lim. 2020. TP-GNN: a graph neural network framework for tier partitioning in monolithic 3D ICs. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.
[24] Wenlong Lyu, Fan Yang, Changhao Yan, Dian Zhou, and Xuan Zeng. 2018. Batch Bayesian Optimization via Multi-objective Acquisition Ensemble for Automated Analog Circuit Design. In *International Conference on Machine Learning*. PMLR, 3306–3314.
[25] Trent McConaghy, Pieter Palmers, Michiel Steyaert, and Georges GE Gielen. 2011. Trustworthy genetic programming-based synthesis of analog circuit topologies using hierarchical domain-specific building blocks. *IEEE Transactions on Evolutionary Computation* 15, 4 (2011), 557–570.

[26] T. McConaghy, P. Palmers, M. Steyaert, and G. G. E. Gielen. 2011. Trustworthy Genetic Programming-Based Synthesis of Analog Circuit Topologies Using Hierarchical Domain-Specific Building Blocks. *IEEE Transactions on Evolutionary Computation* 15, 4 (2011), 557–570.

[27] S Ali Mirsoleimani, Aske Plaat, Jaap Van Den Herik, and Jos Vermaseren. 2015. Parallel monte carlo tree search from multi-core to many-core processors. In *Trustcom/BigDataSE/ISPA*, Vol. 3. IEEE, 77–83.

[28] Paolo Nenzi and Holger Vogt. 2011. Ngspice Users Manual Version 23.

[29] B. Nikolic, E. Alon, and K. Asanovic. 2018. Generating the Next Wave of Custom Silicon. In *ESSCIRC 2018 - IEEE 44th European Solid State Circuits Conference (ESSCIRC)*. 6–11. https://doi.org/10.1109/ESSCIRC.2018.8494310

[30] W. M. Polivka, P. R.K. Chetty, and R. D. Middlebrook. 1980. State-Space Average modelling of converters with parasitics and storage-time modulation. In *1980 IEEE Power Electronics Specialists Conference*. 119–143.

[31] ziga Rojec, Arpad Burmen, and Iztok Fajfar. 2019. Analog circuit topology synthesis by means of evolutionary computation. *Engineering Applications of Artificial Intelligence* 80 (2019), 48–65.

[32] Richard B Segal. 2010. On the scalability of parallel UCT. In *International Conference on Computers and Games*. Springer, 36–47.

[33] Keertana Settaluri, Ameer Haj-Ali, Qijing Huang, Kourosh Hakhamaneshi, and Borivoje Nikolic. 2020. Autockt: Deep reinforcement learning of analog circuit designs. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 490–495.

[34] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.

[35] Ashish Kumar Singh, Kareem Ragab, Mario Lok, Constantine Caramanis, and Michael Orshansky. 2012. Predictable equation-based analog optimization based on explicit capture of modeling error statistics. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 10 (2012), 1485–1498.

[36] Tesla. 2021. Tesla AI Day. https://www.youtube.com/watch?v=j0z4FweCy4M.

[37] Hanrui Wang, Kuan Wang, Jiacheng Yang, Linxiao Shen, Nan Sun, Hae-Seung Lee, and Song Han. 2020. GCN-RL Circuit Designer: Transferable Transistor Sizing with Graph Neural Networks and Reinforcement Learning. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[38] Ye Wang, Michael Orshansky, and Constantine Caramanis. 2014. Enabling efficient analog synthesis by coupling sparse regression and polynomial optimization. In *Proceedings of the 51st Annual Design Automation Conference*. 1–6.

[39] Tong Wu, Zhiqiang Wang, Burak Ozpineci, Madhu Chinthavali, and Steven Campbell. 2019. Automated Heatsink Optimization for Air-Cooled Power Semiconductor Modules. *IEEE Transactions on Power Electronics* 34, 6 (2019), 5027–5031.

[40] Kai-En Yang, Chia-Yu Tsai, Hung-Hao Shen, Chen-Feng Chiang, Feng-Ming Tsai, Chung-An Wang, Yiju Ting, Chia-Shun Yeh, and Chin-Tang Lai. 2020. Fast Design Space Adaptation with Deep Reinforcement Learning for Analog Circuit Sizing. *arXiv preprint arXiv:2009.13772* (2020).

[41] Xiaojian Yang, Maogang Wang, Ryan Kastner, Soheil Ghiasi, and Majid Sarrafzadeh. 2003. Congestion reduction during placement with provably good approximation bound. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 8, 3 (2003), 316–333.

[42] Kazuki Yoshizoe, Akihiro Kishimoto, Tomoyuki Kaneko, Haruhiro Yoshimoto, and Yutaka Ishikawa. 2011. Scalable distributed monte-carlo tree search. In *International Symposium on Combinatorial Search*, Vol. 2.

[43] Guo Zhang, Hao He, and Dina Katabi. 2019. Circuit-GNN: Graph Neural Networks for Distributed Circuit Design. In *International Conference on Machine Learning*. PMLR, 7364–7373.

[44] Shuhan Zhang, Fan Yang, Dian Zhou, and Xuan Zeng. 2020. An efficient asynchronous batch bayesian optimization approach for analog circuit synthesis. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[45] Yi Zhang, Zhongxu Wang, Huai Wang, and Frede Blaabjerg. 2020. Artificial Intelligence-Aided Thermal Model Considering Cross-Coupling Effects. *IEEE Transactions on Power Electronics* 35, 10 (2020), 9998–10002.

[46] Shuai Zhao, Frede Blaabjerg, and Huai Wang. 2021. An Overview of Artificial Intelligence Applications for Power Electronics. *IEEE Transactions on Power Electronics* 36, 4 (2021), 4633–4658.

[47] Z. Zhao and L. Zhang. 2020. An Automated Topology Synthesis Framework for Analog Integrated Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 39, 12 (2020), 4325–4337.

[48] Hongxia Zhou, Chiu-Wing Sham, and Hailong Yao. 2017. Revisiting routability-driven placement for analog and mixed-signal circuits. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 23, 2 (2017), 1–17.

[49] Z. Zhou, S. Belakaria, A. Deshwal, W. Hong, J. R. Doppa, P. Pratim Pande, and D. Heo. 2020. Design of Multi-Output Switched-Capacitor Voltage Regulator via Machine Learning. In *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*. 502–507.