CBTune: Contextual Bandit Tuning for Logic Synthesis

Fangzhou Liu♠ Zehua Pei♠ Ziyang Yu♠ Haisheng Zheng♡ Zhuolun He♠♡ Tinghuan Chen♣ Bei Yu♠

♠CUHK ♡Shanghai AI Lab ♣CUHK (SZ)

Highlights

- We propose **CBTune**, adapting the contextual bandit algorithm to facilitate efficient transformation selection through iterative model tuning.
- We implement the Syn-LinUCB algorithm as the agent and establish a context generator for informed decision-making in the bandit model.
- We present a novel "return-back" mechanism that revisits decisions to avoid local optima, distinguishing it from typical RL scenarios.
- Our method surpasses SOTA approaches for metrics and runtime within the same action space.

Background

ML-Enhanced Synthesis Optimization.

Machine Learning facilitates technology-independent optimization:

1. It models circuit structures to accurately predict performance metrics [4].
2. It employs reinforcement learning for rapid synthesis flow generation in an exponentially large solution space [1].

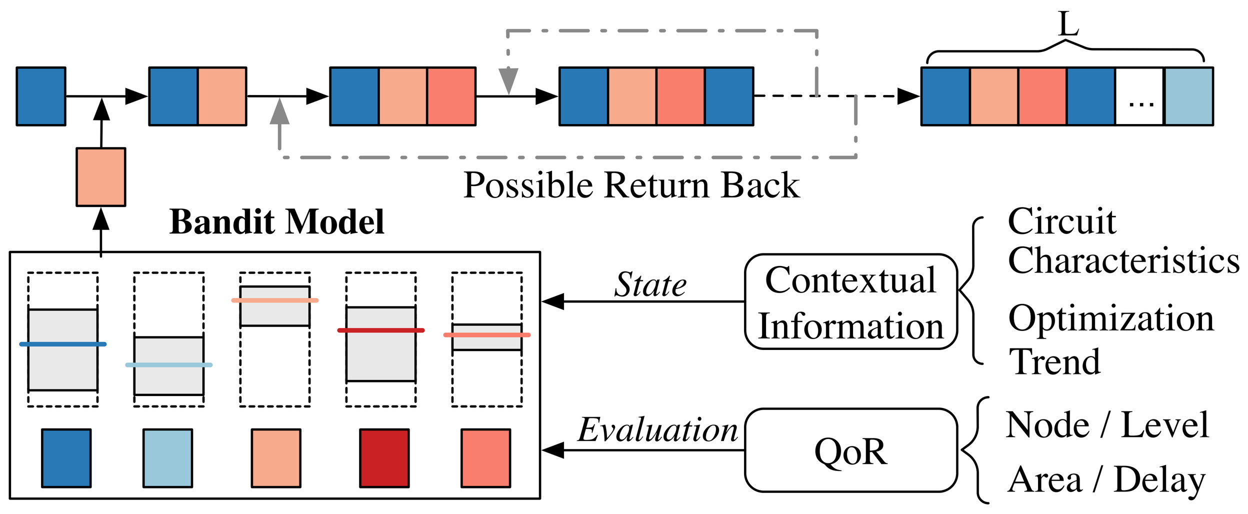

Figure 1. Illustration of Our Proposed Contextual Bandit-Based Approach for Efficient Synthesis Flow Generation.

Bandit-Based Search Model.

The Multi-Arm Bandit (MAB) model, known for its efficiency in generating synthesis flows [3], strikes a balance between exploration and exploitation to optimize rewards. CBTune leverages domain-specific knowledge by integrating contextual data into the MAB model, enabling progressive decision-making depicted in Figure 1.

Motivation

Existing Problems

- NN-based methods are limited by time-consuming dataset preparation and training, as well as restricted transferability and system integration.
- The non-contextual MAB approach neglects key arm features like optimization trends and AIG characteristics. It also makes sequence-based decisions without considering permutations, compromising final performance.

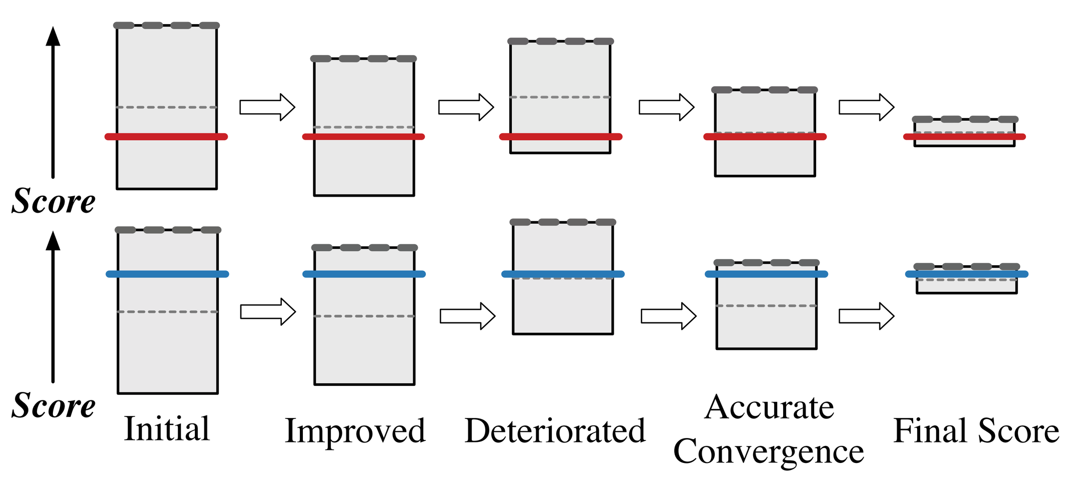

Figure 2. Score Iterations for Each Arm in Bandit Model.

Observations LinUCB [2] improves MAB model by integrating contextual details like arm and environmental features to guide decision-making. The score for each arm a is updated by:

$$\begin{aligned}\text{LinUCB}_a &= E(a|\boldsymbol{x}) + \alpha \text{STD}(a|\boldsymbol{x}) \\ &= \boldsymbol{x}^\top \cdot \boldsymbol{\theta}_a + \alpha\sqrt{\boldsymbol{x}^\top \boldsymbol{A}_a^{-1}\boldsymbol{x}}.\end{aligned} \tag{1}$$

1st term: Estimated Payoff

- Estimates average payoff from $\boldsymbol{x}$
- θ_a represents historical success

2rd term: Upper Confidence Bound

- Controlled by hyperparameter α
- Reflects uncertainty in estimation

Therefore, we propose a tailored bandit model to guide decisions for each individual transformation within the synthesis flow efficiently. This model:

1. Treats each transformation as an "arm" with equal initial UCB scores.
2. Iteratively updates scores to gauge performance.
3. Chooses and refines the highest-scoring arm in each iteration for enhanced score accuracy and reliability.
4. Steers scores towards the arms' true payoffs, with the highest-scoring arm reflecting the best optimization performance.

Pipeline

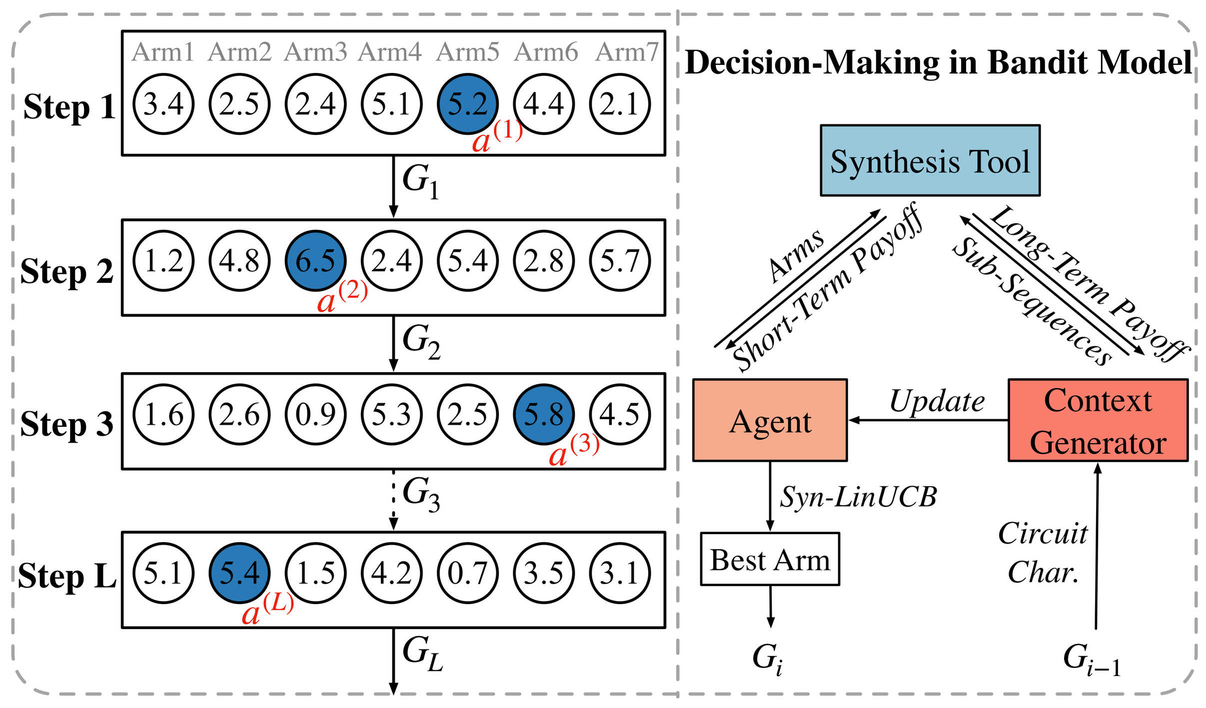

Figure 3. CBTune Framework Overview.

- **Action Space:** $\mathcal{A}$ = {*resub* (*rs*), *resub -z* (*rsz*), *rewrite* (*rw*), *rewrite -z* (*rwz*), *refactor* (*rf*), *refactor -z* (*rfz*), *balance* (*b*)}.
- **Reward** r: the scaled payoff of a single arm execution.

Methodology

Context Generator

The vector $\boldsymbol{x}$, fusing circuit characteristics $\boldsymbol{x^c}$ and the arm's long-term payoff $\boldsymbol{x^l}$, informs the agent's decisions by providing essential environmental and state insights.

Table 1. Contextual Information.

Feature	Example
Circuit Characteristics ($\boldsymbol{x}^c$)	Extracted by **yosys** and **ccirc** #Number of wires/cells/nots, #Maximum delay, #Number of combinational nodes, #Number of high degree comb, #Reconvergence, #Node shape...
Long-term Payoff of the Arm ($\boldsymbol{x}^l$)	Arm: *rewrite* (*rw*); l = 5; m = 1; {**rw**,rf,rf,rw,b} → Nodes: 28010, Level: 66 Arm: *refactor* (*rf*); l = 4; m = 2; {**rf**,b,rf,rw} → Nodes: 28350, Level: 69 {**rf**,rw,b,rs} → Nodes: 28324, Level: 67

Agent (Syn-LinUCB).

1. It utilizes short-term payoffs to direct the agent to select arms toward the optimal target value per step, enhancing local performance.
2. It accounts for long-term payoffs to avert local optima and explore potential optimization trends, fostering improved decision quality.

Algorithm 1 Syn-LinUCB

Input: Arms $a \in \mathcal{A}$, Context weights $\boldsymbol{w} \in \mathbb{R}^d$,
Number of iterations T, Constant ρ.
Output: Best arm a_{best} in this step.

```
r_a ← Reward of all arms;
Extract the AIG characteristics: x_a^c ∈ R^{d_1};
Arm selection times s_a = 0;
for t = 1, 2, ..., T do
    Update the long-term payoff: x_{t,a}^l ∈ R^{d_2};
    Observe features of a ∈ A : x_{t,a} = [x_a^c, x_{t,a}^l] ∈ R^d;
    for ∀a ∈ A do
        Initialize historical context and reward by A_a = I_d, b_a = 0_d, ∀a is new;
        Update hyperparameter α by α = 1.0 + sqrt(log(2.0/ρ) / s_a);
        Update the decision parameter by θ_a = A_a^{-1} b_a;
        Calculate the weighted context x_{t,a}^w = x_{t,a} w;
        Update score by p_{t,a} = θ_a^T (x_{t,a}^w) + α sqrt((x_{t,a}^w)^T A_a^{-1} (x_{t,a}^w));
    end for
    Choose arm by a_t = argmax_{a∈A} p_{t,a};
    Increase the selection count of arm a_t by s_{a_t} = s_{a_t} + 1;
    Update the parameters A_{a_t} and b_{a_t} of the chosen arm a_t by
      A_{a_t} = A_{a_t} + x_{t,a_t} x_{a_t}^T,   b_{a_t} = b_{a_t} + r_a x_{t,a_t};
end for
a_best ← a_t.
```

Return-back Mechanism To amend suboptimal decisions stemming from a lack of historical data, we allow CBTune the capacity to "**regret**" by recording synthesis results in a hash table. This allows CBTune to compare new results with past decisions and, if necessary, return to a crucial step to reselect a better arm, thus improving decision quality.

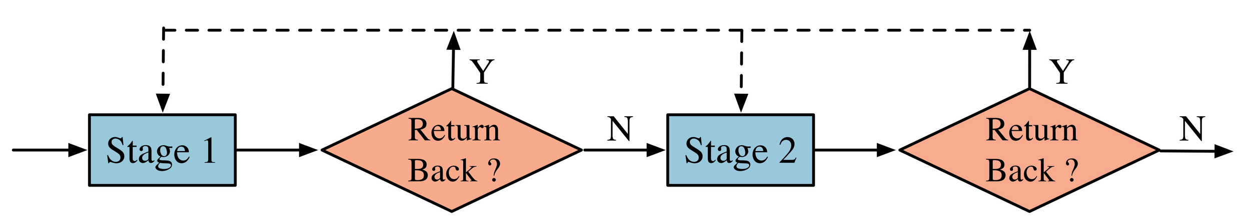

Figure 4. The Return-Back Mechanism in CBTune.

Evaluation Results

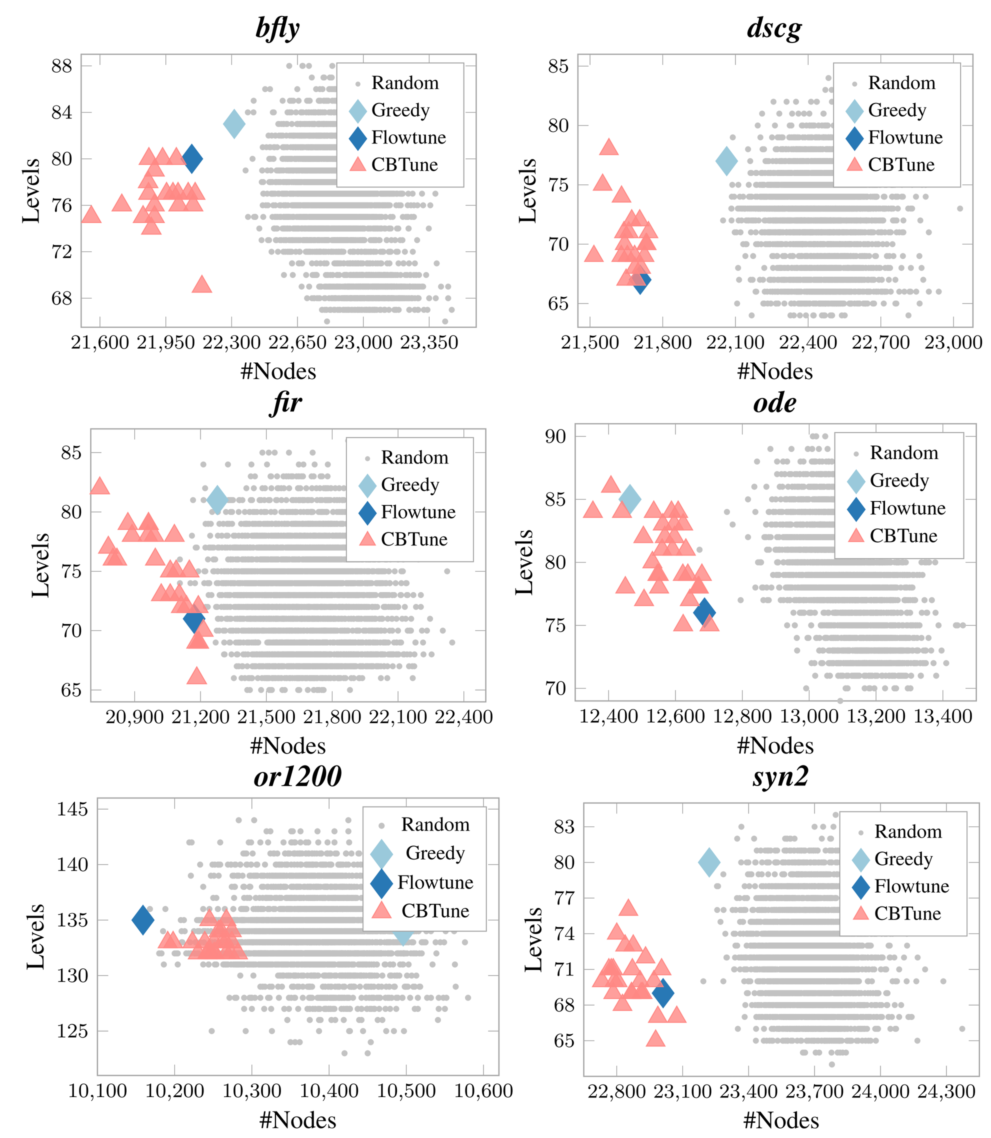

Figure 5. CBTune v.s. FlowTune [3] in AIG Node Optimization.

Benchmark	Initial	Greedy	Flowtune [3]		CBTune		
	#LUTs	#LUTs	#LUTs	$\tau(m)$	$\widehat{\text{\#LUTs}}$	$\overline{\text{\#LUTs}}$	$\tau(m)$
bfly	9019	8269	8216	76.47	**7962**	8086.03	**29.63**
dscg	8534	8313	8302	77.15	**7981**	8119.84	**30.44**
fir	8646	8385	8094	74.23	**7820**	7977.38	**27.6**
ode	5244	5316	5096	34.83	**4920**	5046.71	**17.32**
or1200	2776	2748	2747	20.08	**2731**	2754.07	**15.62**
syn2	8777	8669	8603	81.33	**8234**	8360.53	**31.67**
GEOMEAN	6631.20	6464.69	6364.89	54.04	**6166.39**	6271.82	**24.48**
Ratio Avg.	1.000	0.975	0.960	1.000	**0.930**	0.946	**0.453**

Table 2. CBTune v.s. FlowTune in 6-LUTs Optimization.

Benchmark	Initial	Greedy	DRiLLS [1]		RL4LS*		CBTune	
	#LUTs	#LUTs	$\overline{\text{\#LUTs}}$	$\tau(m)$	$\overline{\text{\#LUTs}}$	$\tau(m)$	$\overline{\text{\#LUTs}}$	$\tau(m)$
max	721	697	694	32.58	687.8	54.34	**684.25**	**6.01**
adder	249	**244**	**244**	24.05	**244**	10.05	**244**	**5.97**
cavlc	116	115	112.2	26.02	111.3	3.22	**111**	**2.37**
ctrl	29	**28**	**28**	24.25	**28**	2.85	**28**	**0.59**
int2float	47	46	42.6	21.7	42.3	2.81	**40**	**2.76**
router	73	67	70.1	22.01	69.5	3.07	**68.11**	**2.32**
priority	264	146	**133.4**	23.32	142.9	5.9	138.86	**3.41**
i2c	353	291	292.1	25.17	289.32	7.55	**283.11**	**3.61**
sin	1444	1451	1441.5	51.15	**1438**	20.1	1441.67	**9.71**
square	3994	3898	3889.4	130	3889	72.88	**3882.11**	**25.99**
sqrt	8084	4807	4708	147.64	4685.3	196.15	**4607**	**36.51**
log2	7584	7660	7583.6	198.6	7580.1	125.28	**7580**	**41.27**
multiplier	5678	5688	5678	180.84	**5672**	187.81	5679.75	**29.08**
voter	2744	1904	1834.7	84.43	**1678.1**	330.48	1682.25	**11.46**
div	23864	4205	7944.4	259.75	7807.1	482	**4180.91**	**25.58**
mem_ctrl	11631	**10144**	10527.6	229.33	10309.7	1985.84	10242.57	**45.81**
GEOMEAN	926.59	732.69	753.49	59.48	748.34	34.54	**712.83**	**8.37**
Ratio Avg.	1.000	0.791	0.813	1.000	0.808	0.581	**0.769**	**0.141**

Table 3. CBTune v.s. NN-enhanced RL in 6-LUTs Optimization.
** Last10 in RL-PPO-Pruned [5].*

Conclusion

- CBTune outperforms FlowTune in both AIG nodes/6-LUT optimization in both metric and runtime. Our method also outshines three RL-based methods by reducing 6-LUT counts up to 4.4%, all achieved in a swift 8.37 minutes per design.
- CBTune efficiently generates synthesis flows with excellent, stable results and fast runtime, without training data or complex procedures.

References

[1] Abdelrahman Hosny, Soheil Hashemi, et al. DRiLLS: Deep reinforcement learning for logic synthesis. pages 581–586, 2020.

[2] Lihong Li, Wei Chu, et al. A contextual-bandit approach to personalized news article recommendation. pages 661–670, 2010.

[3] Walter Lau Neto, Yingjie Li, et al. Flowtune: End-to-end automatic logic optimization exploration via domain-specific multi-armed bandit. 2022.

[4] Nan Wu et al. Lostin: Logic optimization via spatio-temporal information with hybrid graph models. pages 11–18, 2022.

[5] Guanglei Zhou and Jason H Anderson. Area-Driven FPGA Logic Synthesis Using Reinforcement Learning. pages 159–165, 2023.