

DOI: 10.20098/j.cnki.1001-5132.2024.0204

基于 PPO 算法的逻辑综合序列优化通用框架设计

王梦可, 杨朝晖, 查晓婧, 夏银水*

(宁波大学 信息科学与工程学院, 浙江 宁波 315211)

摘要: 逻辑综合通常采用启发式方法将逻辑优化算法组成为序列进行电路性能优化, 而启发式方法难以根据电路和优化目标的差异进行序列自动化调节, 影响了电路优化质量. 为了在集成电路设计中提升序列的自适应生成能力, 将序列优化问题建模为马尔可夫决策过程, 提出一种面向多种逻辑表示的强化学习框架, 利用近端策略优化(Proximal Policy Optimization, PPO)指导智能体来探索序列优化空间, 改善其生成序列的泛化能力. 并将 EPFL 基准电路转变为与-非图(And-Inverter Graph, AIG)和异或多数图(Xor-Majority Graph, XMG)形式, 分别经由所提出的框架进行实验, AIG 形式下本文方法与 DRiLLS 和 BOiLS 方法相比分别有 18.66 百分点和 27.67 百分点的性能提升; XMG 形式下则可提升原始电路性能约 37.34%. 实验结果表明, 由本文方法生成的算法序列对电路性能有较大改进.

关键词: 逻辑综合; 序列优化; 强化学习; 近端策略优化

中图分类号: TP331.2; TN431.2 **文献标志码:** A **文章编号:** 1001-5132 (2025) 02-0078-08

电子设计自动化(Electronic Design Automation, EDA)是集成电路芯片设计的重要组成部分, 它是利用计算机辅助软件设计来完成芯片功能设计、综合、验证、物理设计等流程的设计方式^[1]. 逻辑综合将高级电路语言转换成含有工艺信息的门级网表, 包括翻译、优化和工艺映射三个主要阶段. 逻辑综合中每个阶段的操作都是为了在有限时间内提高最终网表的性能, 其中优化是在满足设计约束前提下经布尔代数理论来改进功耗、延时或者面积, 且优化阶段也并不是一个优化算子就能完成, 需要一系列综合优化算子的排列组合共同实现.

研究者们以“序列”描述综合优化算子的组合顺序, 通常采用启发式算法获取固定序列来优化电路的结果质量(Quality of Results, QoR), 如延时和面积. 但传统启发式算法获取的优化序列无法根据电路结构进行调节, 导致优化质量不佳, 泛化性较低^[2].

机器学习的发展为集成电路设计提供了新思路, 基于机器学习算法的电路优化方法已经应用到包括高级综合^[3]、逻辑综合^[4]、布局布线^[5]等 EDA 流程的各个环节. 文献[6]提出使用卷积神经网络(Convolutional Neural Network, CNN)对随机生成的优化序列进行分类, 将序列优化问题转变为分类问题. 然而该方法生成足够多数据集时, 需要花费大量时间, 且针对某一电路效果好的序列并不一定适用于所有电路, 因此该框架缺乏通用性. 文献[7]通过图神经网络(Graph Neural Network, GNN)来提高电路表达能力, 在延时约束下优化电路面积. 这种基于 GNN 强化学习的方法同样需耗费大量时间用于 GNN 训练, 增加了设计人员的电路优化成本.

鉴于启发式算法难以面向不同电路生成优质序列, 而机器学习算法利用神经网络进行分类需要大量数据集, 且 GNN 训练十分耗时. 因此本文提

收稿日期: 2024-02-18.

宁波大学学报(理工版)网址: <http://journalg.nbu.edu.cn/>

基金项目: 国家自然科学基金(62131010, U22A2013); 国家自然科学基金青年项目(62304115); 浙江省自然科学基金创新群体课题(LDT23F04021F04); 浙江省科研计划一般项目(Y202248965).

第一作者: 王梦可, 硕士研究生, 主要研究方向: 集成电路电子设计自动化. E-mail: 2111082020@nbu.edu.cn

*通信作者: 夏银水, 教授, 主要研究方向: 低功耗集成电路设计及电子设计自动化. E-mail: xiayinshui@nbu.edu.cn

出一种面向多种逻辑表示的强化学习框架, 利用近端策略优化(Proximal Policy Optimization, PPO)^[8]指导智能体来探索优化序列空间以及电路的 QoR.

1 背景介绍

1.1 问题描述

开源逻辑综合工具 ABC^[9]、ALSO(Advanced Logic Synthesis and Optimization)^[10]主张将逻辑电路表示为与-非图(And-Inverter Graph, AIG)^[11]、多数-非图(Majority-Inverter Graph, MIG)^[12]、异或多数图(Xor-Majority Graph, XMG)^[13]等形式的门级网表, 并根据不同逻辑表示提出了多种优化算子, 例如针对 AIG 有 rewrite、refactor、balance 等. AIG 优化算子间组合顺序的差异会引起 AIG 电路结构的变化, 导致工艺映射后延时和面积的不同. 面对 k 种优化算子设计长度为 m 的序列, 存在 k^m 种序列组合方式, 增加优化算子会导致指数级的探索空间, 影响优化序列的搜索难度. 因此, 面向特定逻辑表示, 如何快速地获取一组能够显著减小电路 QoR 的优化算子序列成为我们的研究目标.

1.2 强化学习

强化学习是指智能体在与环境交互中不断学习, 完成特定目标的一种方法. 智能体通过感知外界环境的状态和奖励进行学习和决策, 决策在动作环境中选取合适动作, 智能体执行动作又会影响环境, 环境反馈给智能体相应的奖励. 文中环境表述为状态空间 S ; 可选择动作表示为动作空间 A ; 智能体根据当前环境 s_t 决定动作 a_t 的过程由策略函数 $\pi_\theta(a_t|s_t)$ 表示; 当执行动作 a_t 后, 当前环境转变为新环境 s_{t+1} 的概率用转移函数 $P(s_{t+1}|s_t, a_t)$ 表示; 同步获取环境反馈的奖励表示为奖励函数 $R_t(s_t, a_t, s_{t+1})$. 从 t 时刻获得的累积奖励被定义为 $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$, 其中, γ 为 0~1 之间的折扣因子, r_t 为在 t 时刻获得的奖励. 强化学习的目标是找到一个策略, 使得智能体在一段时间内获得的累积奖励最大化.

1.3 AIG 和 XMG

基于布尔逻辑的图形化表示是将电路抽象为只含节点和连接边的有向无环图, 根据节点逻辑功能, 逻辑表示形式可分为 AIG、XMG 等. AIG 节点表示逻辑“AND”运算, 连接边分为常规边和补

边, 其中补边表示连接有反相器; XMG 中的节点由多数表决运算“MAJ”和三输入异或运算“XOR”组成, 其中“MAJ”用逻辑表达式表示为 $M(x, y, z) = xy + xz + yz$, “XOR”表示为 $R(x, y, z) = x \oplus y \oplus z$.

图 1 是全加器的 AIG 和 XMG 表示形式, 其中 C_{out} 表示进位输出, S_{sum} 描述求和输出. 从图中可以看出, AIG 节点数为 8, 层级为 3; XMG 节点数为 2, 层级为 1. 与 AIG 相比, XMG 提供一种更紧凑的逻辑表示方式, 节点数和层级都更少. 表 1 为常用基准电路 AIG 和 XMG 表示下的节点数和层级对比, 从表中数据可以看出, XMG 可以在节点数或层级上产生比 AIG 更优的结果. 在逻辑综合优化中, 可通过优化算子改变 AIG 或 XMG 结构来改善电路性能. 此外, 节点的逻辑功能差异将导致优化算子无法适用所有形式, 因此需要设计通用架构, 在面向不同优化算子时, 通过探索逻辑表示方式的异同, 自适应生成优化序列, 按序执行序列中的算子, 从而优化电路的 QoR.

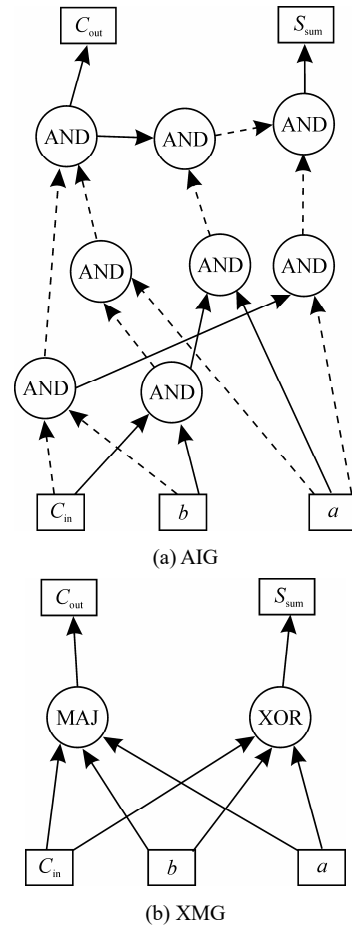


图 1 全加器的图形化表示形式

表1 AIG与XMG节点数和层级比较

电路	AIG		XMG	
	节点数	层级	节点数	层级
adder	1 020	255	383	128
max	2 865	287	2 151	293
multiplier	27 062	274	15 966	135
sin	5 416	225	3 895	150
square	18 484	250	12 780	152
control	174	10	133	8
cavlc	693	16	717	16
i2c	1 342	20	1 314	17
int to float	260	16	250	15
memory	46 836	114	44 794	114
priority	978	250	915	126
voter	13 758	70	7 583	62

2 强化学习算法框架

强化学习算法框架主要由两个部分组成: 逻辑综合环境和强化学习环境(即智能体), 图2描述了两个部分的交互过程. 首先从逻辑综合环境的ABC和YOSYS(Yosys Open Synthesis Suite)^[14]或ALSO和YOSYS中进行电路状态提取, 传递给智能体, 对应图2中“状态表示”; 智能体根据神经网络的策略函数从逻辑综合环境的动作空间中选择优化算子, 对应图2中“动作选择”; 在ABC或ALSO中执行动作得到当前的QoR, 计算后作为奖励传递给智能体, 对应图2中“奖励计算”, 智能体通过奖励函数判断此次动作的优劣. 通过不断

交互进行智能体的训练, 从而在限定序列长度下输出合适的优化算子序列来实现电路QoR的优化.

2.1 逻辑综合环境

2.1.1 状态表示

状态作为强化学习环境的输入, 对智能体的策略学习至关重要. 合理的状态表示可以让智能体感知到当前环境信息和因执行动作产生的变化, 即感知到当前电路状态. 为了状态表示更具有泛化性, 将状态特征进行归一化处理, 从而消除智能体受逻辑表示和电路结构差异对状态感知的影响. 在逻辑综合环境中, 通过ABC和YOSYS提取状态, 然后传递给强化学习环境进行处理. 分析AIG与XMG的共同点, 由于可以表达信息的只有节点和连接边, 因此, 表达各种逻辑运算的节点数量和执行优化算子前后节点数量的变化情况可以作为状态信息, 电路结构变化还可以通过层级来表示, 序列长度也可以作为状态信息加入其中. 因此, AIG的状态空间包含以下8个信息:

$$S_{\text{AIG}} = \left\{ \begin{array}{l} \text{当前节点数/初始节点数,} \\ \text{当前层级数/初始层级数,} \\ \text{上一命令节点数/初始节点数,} \\ \text{上一命令层级数/初始层级数,} \\ \text{与门数/去掉输入和输出的节点数,} \\ \text{非门数/去掉输入和输出的节点数,} \\ \text{或门数/去掉输入和输出的节点数,} \\ \text{当前序列长度/序列总长度.} \end{array} \right.$$

XMG状态空间与AIG类似, 区别在于XMG

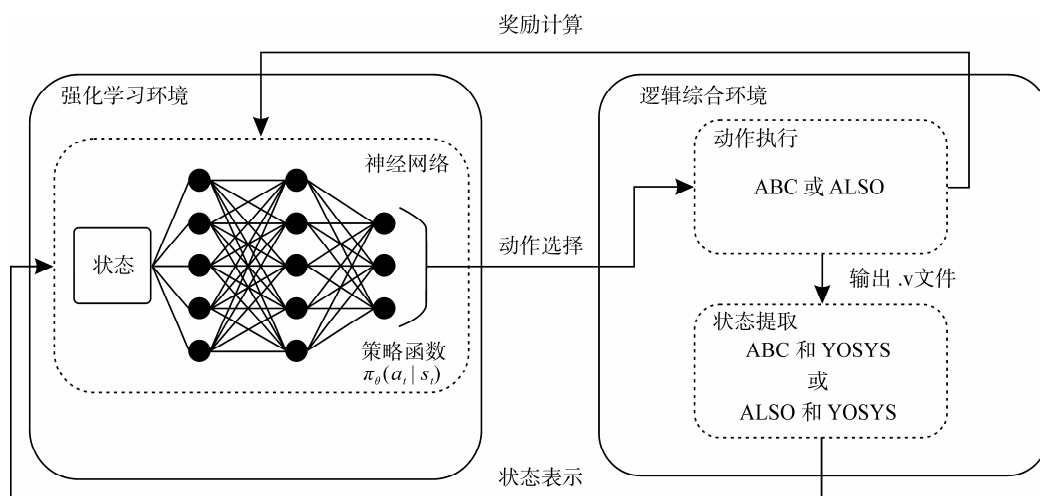


图2 强化学习算法整体框架

中节点逻辑功能不同. XMG 的状态空间包含以下 7 个信息:

$$S_{\text{XMG}} = \begin{cases} \text{当前节点数/初始节点数,} \\ \text{当前层级数/初始层级数,} \\ \text{上一命令节点数/初始节点数,} \\ \text{上一命令层级数/初始层级数,} \\ \text{MAJ 门数/去掉输入和输出的节点数,} \\ \text{XOR 门数/去掉输入和输出的节点数,} \\ \text{当前序列长度/序列总长度.} \end{cases}$$

2.1.2 动作空间

智能体会根据策略选择相应的动作传递到逻辑综合环境. 动作空间的大小决定智能体的探索空间, 丰富的动作空间设计能让智能体有足够的探索空间有机会获取更好的解, 但过大的动作空间也会让探索变得困难. 假设优化序列长度 $k=10$, 动作空间大小 $n=5$, 探索空间大小为 5^{10} (即 n^k), 当动作空间 n 增大时, 会让指数级的探索空间爆发性增长. 因此需要对动作空间进行精心设计.

AIG 通过与 ABC 交互来实现序列优化, 其动作空间是 ABC 中的优化算子. 之前大多数工作仅使用针对 AIG 的基础优化算子 (如 rewrite、refactor) 进行逻辑优化. 由文献[15]可知 LUT 优化能降低逻辑网络复杂度, 可用于优化 AIG, 因此可在 AIG 动作空间中加入 LUT 优化算子 (如 blut) 为 AIG 序列优化提供更多可能性. AIG 动作空间设计为: $A_{\text{AIG}} = \{\text{rewrite; rewrite -z; resub -K16; resub -K16 -N2; refactor; refactor -z; balance; dch balance; dc2; ifraig; sopb; blut; dsdb}\}$. XMG 电路的序列优化通过与 ALSO 交互来实现, 其动作空间是 ALSO 中的优化算子. XMG 的动作空间设计为: $A_{\text{XMG}} = \{\text{xmgban; xmgrs; xmgrw; xmgrw -a; xmgrw --only_maj; xmgrw --xor3_flatten}\}$.

2.1.3 奖励函数

奖励函数是指导智能体行为的重要指标. 执行动作后, 奖励函数将环境的反馈转换为奖励值, 指导智能体进行下一步决策, 因此奖励函数的设计应满足奖励预期与学习目标一致的要求. 为了让智能体能够正确认识到当前优化序列的质量好坏, 奖励函数设计都应基于该事实. 假设长度为 10 的序列, 如果只将每个优化算子执行的结果与上个算子执行的结果作比较, 若优化效果变好记为 1,

变差记为 -1, 当第 1 个优化算子将电路恶化到无法弥补的地步, 后面 9 个算子都在变好, 但效果仍不如初始电路时, 序列整体奖励值为 8, 但结果是奖励值与目标明显不符. 因此, 在设计奖励函数时, 首先应考虑当前效果与原始效果的差异, 通过当前效果与前一个算子执行效果的比值来进行奖励值 (R_t) 设计, 具体公式为:

$$R_t = \begin{cases} \frac{Q_{t-1} - Q_t}{Q_{t-1}}, Q_t < Q_{\text{initial}}, \\ -1 + \frac{Q_{t-1} - Q_t}{Q_{t-1}}, Q_t \geq Q_{\text{initial}}, \end{cases} \quad (1)$$

式中: Q_t 为当前优化算子执行后的 QoR; Q_{t-1} 为上个优化算子执行后的 QoR; Q_{initial} 为原始电路结果. 比较 Q_t 与 Q_{initial} , 若 Q_t 比 Q_{initial} 小, 说明当前优化算子执行后的 QoR 比原始电路结果更差, 给一个负奖励, 否则给出正奖励, 具体数值通过 Q_t 与 Q_{t-1} 计算得出.

2.2 强化学习环境

PPO 是一种用于强化学习的策略梯度方法, 它具有稳定性强、计算效率高和算法复杂度低等优势. 策略梯度算法工作原理是将策略表示成与奖励有关的连续函数, 使用梯度上升更新策略进行梯度优化, 目标是最大化策略函数. 策略网络以状态 s_t 作为输入, 输出是从优化算子中选出动作 a_t . 通过在环境中运行策略以获得策略损失 $L(\theta)$ 及其梯度的样本, 策略梯度的目标函数为:

$$L(\theta) = E_t \left[\log \pi_{\theta}(a_t | s_t) \hat{A}_t \right], \quad (2)$$

式中: $\pi_{\theta}(a_t | s_t)$ 是在时间步长 t 内, 在状态 s_t 选择动作 a_t 时, 以 θ 为参数的策略; E_t 为有限轨迹在一定时间范围内的期望; $\hat{A}_t$ 为优势函数. 使用策略比率 r_t 来评估更新后策略与前一个策略在时间步长 t 处的相似度, 定义如下:

$$r_t(\theta) = \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta, \text{old}}(a_t | s_t)}. \quad (3)$$

较大的 $r_t(\theta)$ 值表示更新后的策略与旧策略差异过大. 策略变动过大可能导致训练不稳定, 因此, PPO 使用截断函数将策略更新限制在一个合理的范围内, 目标函数变为:

$$L_{\text{clip}}(\theta) = E_t \left[\min(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), \right.$$

$$1 - \varepsilon, 1 + \varepsilon) \hat{A}_t \Big], \quad (4)$$

式中: $\text{clip}(r_t(\theta), 1 - \varepsilon, 1 + \varepsilon)$ 表示将 $r_t(\theta)$ 限制在 $[1 - \varepsilon, 1 + \varepsilon]$ 内; $\min$ 是最小化操作. 上式可修改为:

$$L_{\text{clip}}(\theta) = \begin{cases} (1 + \varepsilon) \hat{A}_t, & \hat{A}_t > 0, r_t(\theta) > 1 + \varepsilon, \\ (1 - \varepsilon) \hat{A}_t, & \hat{A}_t < 0, r_t(\theta) < 1 - \varepsilon, \\ L_{\text{clip}} = r_t(\theta) \hat{A}_t, & \text{其他.} \end{cases} \quad (5)$$

当 $\hat{A}_t$ 大于 0 时, $L_{\text{clip}}(\theta)$ 与 $r_t(\theta)$ 成正比, clip 操作和 $\min$ 操作会将 $r_t(\theta)$ 限制到 $1 + \varepsilon$, 防止策略更新幅度过大而错过当前最佳优化算子, 影响最终优化结果. 同样地, 对于 $\hat{A}_t$ 小于 0, $L_{\text{clip}}(\theta)$ 与 $r_t(\theta)$ 成反比, clip 操作和 $\min$ 操作会将 $r_t(\theta)$ 限制到 $1 - \varepsilon$, 防止更新幅度太小陷入局部最优. 优势函数为:

$$\hat{A}_t = \sum_t \gamma^t r_t - V(s_t), \quad (6)$$

式中: r_t 为每一步所获得的奖励; γ 为折扣因子; $V(s_t)$ 为状态值函数.

2.3 算法实现

本文改进强化学习算法执行的伪代码如下:

输入: 基准电路;

输出: 优化序列;

```

1  整体环境初始化;
2   $Q_{\text{initial}} = D_{\text{delay,initial}} \times A_{\text{area,initial}}$ ;
3  for  $i = 1$  to  $E$  do
4    逻辑综合环境初始化;
5    for  $t = 1$  to  $L$  do
6      根据策略  $\pi_\theta(a_t | s_t)$  选择动作  $a_t$ ;
7      执行动作获取下一状态  $s_{t+1}$ ;
8       $Q_t = D_{\text{delay},t} \times A_{\text{area},t}$ ;
9      根据式(1)得到当前奖励  $R_t$ ;
10   end
11   根据式(6)评估优势值;
12   根据式(5)更新策略  $\pi_\theta(a_t | s_t)$ ;
13   使用 adam 优化器更新神经网络;
14 end
```

算法中输入为基准电路, 输出为优化序列. 第 1 行进行整体环境初始化; 第 2 行计算初始状态电路的 QoR, 文中用电路延时和面积的乘积 Q_{initial} 表示. 第 3~14 行表示每次迭代的具体实现, 其中第 4

行表示每次迭代开始首先进行逻辑综合环境的初始化, 第 5~10 行表示一次迭代中选择并执行动作获得奖励的过程, 智能体根据策略 $\pi_\theta(a_t | s_t)$ 选择动作 a_t 并执行相应动作后, 获取下一状态 s_{t+1} , 计算执行动作 a_t 后的 Q_t 以获取当前奖励 R_t . 第 11~13 行表示一次迭代结束后根据式(6)来评估优势值, 根据式(5)更新策略 $\pi_\theta(a_t | s_t)$, 同时更新神经网络进入下一次迭代.

3 实验结果分析

3.1 实验设置

本文提出的优化算法在 Python v3.7.16 中实现, 并使用 PyTorch v1.13.1^[16]来训练神经网络, 实验在 2.40 GHz 的英特尔至强 4210R CPU 和英伟达 RTX 3090 上进行. 通过 ABC 和 ALSO 分别搭建 AIG 和 XMG 的逻辑综合框架, 并在 EPFL 基准电路^[17]进行实验.

实验参数设置如下:

迭代次数(E): 50 次;

序列长度(L): 20;

adam 优化器步长: 0.01;

clip 参数 ε : 0.1;

折扣因子 γ : 0.98.

3.2 实验结果

分别基于 AIG 和 XMG 表示形式, 利用本文提出方法自动化生成一组优化序列并获得 FPGA 映射结果. 为保证实验参数一致性, 在迭代次数与序列长度相同条件下与基于强化学习的 DRiLLS^[18]和基于贝叶斯优化的 BOiLS^[19]方法相比较. 以执行 ABC 常用序列优化脚本 resyn2^[20]的结果作为基准, 比较不同方法对电路 QoR 的提升程度.

3.2.1 AIG 结果分析

表 2 为 EPFL 基准电路上优化 AIG 的实验结果, 为了更直观地看出各种方法的优化效果, 以式(7)方式进行计算:

$$P = \frac{Q_s - Q_m}{Q_s} \times 100\%, \quad (7)$$

式中: P 表示优化程度; Q_s 表示基准电路的实验结果; Q_m 表示优化方法的实验结果. 表 2 中所有实验数据均为 ABC 中执行 LUT-6 工艺映射后的结果, 以 12 个基准电路的 QoR 平均值作为最终的实

表2 本文方法的 AIG 形式与其他方法结果比较

基准电路	resyn2 基准			DRiLLS ^[16]			BOiLS ^[17]			本文		
	面积	延时	QoR	面积	延时	QoR	面积	延时	QoR	面积	延时	QoR
adder	257	51	13 107	276	51	14 076	281	51	14 331	241	51	12 291
max	777	41	31 857	764	35	26 740	789	39	30 771	884	10	8 840
multiplier	5 916	53	313 548	6 241	51	318 291	6 458	44	284 152	6 359	27	171 693
sin	1 452	36	52 272	1 740	33	57 420	1 769	35	61 915	1 538	31	47 678
square	3 948	50	197 400	3 861	19	73 359	3 926	47	184 522	4 055	11	44 605
control	28	2	56	28	2	56	27	2	54	27	2	54
cavlc	120	4	480	116	4	464	121	4	484	132	3	396
i2c	307	4	1 228	282	3	846	258	4	1032	275	3	825
int to float	48	3	1 44	41	3	123	42	3	126	43	3	129
memory	11 492	25	287 300	5 505	11	60 555	5 687	10	56 870	9 428	10	94 280
priority	182	27	4 914	137	15	2 055	149	7	1 043	148	4	592
voter	1 785	13	23 205	1 475	12	17 700	1 520	13	19 760	1 469	12	17 628
平均结果	—	—	77 125.92	—	—	47 640.42	—	—	54 588.33	—	—	33 250.92
P/%	—	—	—	—	—	38.23	—	—	29.22	—	—	56.89

注: 粗体表示 QoR 的最好结果。

验结果进行比较, 表2中用“平均结果”进行表述。与 resyn2 基准方法相比, DRiLLS、BOiLS 和本文方法的 QoR 分别提升 38.23%、29.22% 和 56.89%, 相比 DRiLLS 和 BOiLS 方法, 本文方法的 QoR 分别提升了 18.66 个百分点和 27.67 个百分点。这是因为 DRiLLS 虽然采用强化学习方法, 但其状态空间设计仅包含当前 AIG 信息, 无法学习与历史状态相关的信息, 策略更新不及时, 影响智能体的学习效率。从表2结果中还可以看出, DRiLLS 方法在部分电路中提升较大, 但部分电路优化效果提升不明显, 甚至会变得更差, 主要因为采用的演员-评论家策略函数(Actor-Critic Algorithm)不够稳定。BOiLS 采用基于高斯优化的贝叶斯优化算法, 在处理序列优化此类离散问题时, 算法采样和优化过程都会比较困难, 计算复杂度高, 导致在一定迭代次数内序列的优化效果不明显, 使得电路 QoR 效果不理想。本文方法对大部分电路的 QoR 提升效果较好, 虽然个别电路提升效果较差, 甚至其中 memory 电路优化程度还不如其他方法, 这是因为本文新加入的优化算子对个别节点数过多的电路优化效果不够明显, 还需对优化算子选择做进一步研究。

3.2.2 XMG 结果分析

由于目前针对 XMG 的优化算法研究较少, 本

文方法在 XMG 形式上与原始电路进行比较, 表3为 EPFL 基准电路上优化 XMG 的实验结果, “平均结果”一行表示12个基准电路的 QoR 平均值。从表3可以看出, 与原始电路相比, 本文所提出方法在12个基准电路上的优化结果提升 37.34%。首先, 本文方法在设置状态空间时不只是针对单一电路表示形式, 而是从电路图形化表示的共同点出发, 选取更全面的表示方式, 从而使智能体能够更准确地感知到环境信息。其次, 本文奖励函数从总体效果考虑, 根据动作效果设置更合理的奖励值, 指导智能体识别动作的优劣。最后, PPO 策略为智能体的训练提供稳定性。因此本文方法面向 XMG 形式电路时, 在同样迭代 50 次情况下也取得了不错的效果。

4 结语

为解决启发式算法在逻辑综合电路优化上效果不理想的问题, 本文提出一种面向多种逻辑表示的强化学习框架, 使其支持 AIG 和 XMG 形式的电路优化, 并提高了其生成序列的泛用性。采用 PPO 算法, 通过捕获 AIG 或 XMG 中的关键信息来丰富状态空间; 融入 LUT 优化算子和组合优化算子来更新动作空间; 设置与优化目标一致的奖励

表3 本文方法的 XMG 形式与原始结果比较

基准电路	原始			本文		
	面积	延时	QoR	面积	延时	QoR
adder	255	127	32 385	645	9	5 805
max	1 022	208	212 576	1 397	96	134 112
multiplier	7 203	129	929 187	8 700	67	582 900
sin	1 882	98	184 436	2 121	83	176 043
square	6 379	124	790 996	8 226	20	164 520
control	54	3	162	49	4	196
cavlc	291	9	2 619	343	7	2 401
i2c	526	8	4 208	538	6	3 228
int to float	94	7	658	95	6	570
memory	18 281	69	1 261 389	19 000	57	1 083 000
priority	329	63	20 727	328	62	20 336
voter	4 104	37	151 848	3 003	24	72 072
平均结果	—	—	256 513.64	—	—	160 738.79
P/%	—	—	—	—	—	37.34

函数,为智能体提供明确的指导;PPO 算法能够提高策略稳定性,显著提高序列优化效果的通用性.本文算法具有稳定性强、适应性广泛等优点,可以在有限的迭代次数中提升电路的结果质量.

参考文献:

- [1] 储著飞,王伦耀,夏银水.基于多逻辑域的逻辑综合研究进展[J].微纳电子与智能制造,2021,3(2):64-73.
- [2] LI X, CHEN L, YANG F, et al. HlMap: a heuristic and iterative logic synthesis approach[C]//Proceedings of the 59th ACM/IEEE Design Automation Conference. San Francisco, CA, USA. ACM, 2022:415-420.
- [3] CHEN H Z, SHEN M H. A deep-reinforcement-learning-based scheduler for high-level synthesis[C]//Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. Seaside, CA, USA. ACM, 2019:117.
- [4] NETO W L, LI Y, GAILLARDON P E, et al. Flowtune: end-to-end automatic logic optimization exploration via domain-specific multi-armed bandit[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2022, 42(6):1912-1925.
- [5] AGNESINA A, CHANG K, LIM S K. VLSI placement parameter optimization using deep reinforcement learning [C]//2020 IEEE/ACM International Conference on Computer Aided Design (ICCAD). San Diego, CA, USA. IEEE, 2020:1-9.
- [6] YU C X, XIAO H P, DE MICHELI G. Developing synthesis flows without human knowledge[C]//2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC). San Francisco, CA, USA. IEEE, 2018:1-6.
- [7] YANG C H, XIA Y S, CHU Z F, et al. Logic synthesis optimization sequence tuning using RL-based LSTM and graph isomorphism network[J]. IEEE Transactions on Circuits and Systems II: Express Briefs, 2021, 69(8): 3600-3604.
- [8] GU Y, CHENG Y H, CHEN C L P, et al. Proximal policy optimization with policy feedback[J]. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2022, 52(7): 4600-4610.
- [9] BRAYTON R, MISHCHENKO A. ABC: an academic industrial-strength verification tool[C]//International Conference on Computer Aided Verification. Berlin, Heidelberg: Springer, 2010:24-40.
- [10] CHU Z. Advanced logic synthesis and optimization tool (ALSO) (Release 20190221)[EB/OL]. [2024-02-18]. <https://gitee.com/zfchu/also>.
- [11] MISHCHENKO A, CHATTERJEE S, BRAYTON R. DAG-aware AIG rewriting a fresh look at combinational logic synthesis[C]//2006 43rd ACM/IEEE Design Automation Conference. San Francisco, CA, USA. IEEE, 2006:532-535.
- [12] AMARÚ L, GAILLARDON P E, DE MICHELI G. Majority-inverter graph: a novel data-structure and

- algorithms for efficient logic optimization[C]//2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC). San Francisco, CA, USA. IEEE, 2014:1-6.
- [13] CHU Z F, SOEKEN M, XIA Y S, et al. Structural rewriting in XOR-majority graphs[C]//Proceedings of the 24th Asia and South Pacific Design Automation Conference. Tokyo, Japan. ACM, 2019:663-668.
- [14] SHAH D, HUNG E, WOLF C, et al. Yosys+ nextpnr: an open source framework from verilog to bitstream for commercial FPGAs[C]//2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). San Diego, CA, USA. IEEE, 2019:1-4.
- [15] AMARÚ L, POSSANI V, TESTA E, et al. LUT-based optimization for ASIC design flow[C]//2021 58th ACM/IEEE Design Automation Conference (DAC). San Francisco, CA, USA. IEEE, 2021:871-876.
- [16] PASZKE A, GROSS S, MASSA F, et al. PyTorch: an imperative style, high-performance deep learning library [EB/OL]. [2024-01-12]. <http://arxiv.org/abs/1912.01703>.
- [17] AMARÚ L, GAILLARDON P E, DE MICHELI G. The EPFL combinational benchmark suite[C]//Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS). 2015:1-5.
- [18] HOSNY A, HASHEMI S, SHALAN M, et al. DRiLLS: deep reinforcement learning for logic synthesis[C]//2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC). Beijing, China. IEEE, 2020:581-586.
- [19] GROSNI A, MALHERBE C, TUTUNOV R, et al. BOiLS: Bayesian optimisation for logic synthesis [C]//2022 Design, Automation & Test in Europe Conference & Exhibition (DATE). Antwerp, Belgium. IEEE, 2022:1193-1196.
- [20] MISHCHENKO A, CHATTERJEE S, BRAYTON R. DAG-aware AIG rewriting: a fresh look at combinational logic synthesis[C]//2006 43rd ACM/IEEE Design Automation Conference. San Francisco, CA, USA. IEEE, 2006:532-535.

PPO-based universal framework design for logic synthesis sequence optimization

WANG Mengke, YANG Zhaohui, ZHA Xiaojing, XIA Yinshui

(Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo 315211, China)

Abstract: Logic synthesis typically employs heuristic methods to compose a sequence of logic optimization algorithms for circuit performance improvement. However, heuristic methods face challenges in automatically adjusting sequences based on circuit and optimization objectives, thus affecting the quality of circuit optimization. In order to enhance the capability of adaptive sequence generation in integrated circuit design, this paper models the sequence optimization problem as a Markov Decision Process and proposes a reinforcement learning framework for multiple logic representations. The framework utilizes Proximal Policy Optimization (PPO) to guide the agent in exploring the sequence optimization space and improve the generalization ability of its sequence generation. The paper transforms EPFL benchmark circuits into AIG and XMG forms, respectively, and conducts experiments on them by using the proposed framework. Compared to DRiLLS and BOiLS methods, the proposed method achieves performance improvements of 18.66 percentages and 27.67 percentages in the AIG form, and approximately 37.34% improvement in the original circuit performance in the XMG form. The experimental results demonstrate significant improvements in circuit performance achieved by the algorithm sequences generated with the proposed method.

Key words: logic synthesis; sequence optimization; reinforcement learning; proximal policy optimization

(责任编辑 章践立)