arXiv:2303.02290v1 [cs.AI] 4 Mar 2023

面向更高覆盖率的冲突驱动结构学习自动测试图形生成

Hui-Ling Zhen[1], Naixing Wang[2], Junhua Huang[1],Xinyue Huang[2],Mingxuan Yuan[1] 和 Yu Huang[2]
1. 诺亚方舟实验室，华为；2. 海思，华为

摘要—**随着集成电路设计复杂度不断提高所带来的挑战日益加剧，布尔可满足性（SAT）已成为结构化自动测试图形生成技术的有力替代方案。然而，将电路测试问题转换为合取范式（CNF）的高昂代价限制了 SAT 在工业自动测试图形生成场景中的应用，导致测试覆盖率下降。为解决这一问题，本文首先提出了一种冲突驱动结构学习（CDSL）自动测试图形生成算法，将现代 SAT 求解器中的冲突驱动启发式方法直接应用于故障传播与激活的逻辑锥。所提出的 CDSL 算法由三个部分组成：（1）根据蕴含图学习各种冲突约束，以剪枝搜索空间。（2）应用冲突驱动蕴含和正当化，以提高决策准确性和求解效率。（3）针对低覆盖率调试，进一步提出一种基于冲突的诊断方法，通过放宽或修改主输入上的部分约束，使中止故障变得可测试。在工业电路上的大量实验结果表明，所提出的 CDSL 算法具有良好的有效性和效率。结果显示，与基于 SAT 的自动测试向量生成相比，所提出的 CDSL 平均可将中止故障减少 25.6%，同时运行时间减少94.51%。在两阶段计算流程下，结果表明，与单阶段结构化算法相比，所提出的 CDSL 可将中止故障进一步减少46.37%，并使故障覆盖率提高3.19%。此外，冲突诊断平均可将中止故障减少8.89%，并使故障覆盖率提高0.271%。**

索引词—**冲突驱动、自动测试图形生成、冲突诊断**

I. 引言

器件尺寸不断减小、设计复杂度持续提高，使得市场对高产品质量和低百万分率缺陷数（DPPM）目标的需求日益增长。因此，基于扫描的结构型测试变得比以往任何时候都更加重要，而自动测试图形生成（ATPG）也已成为生成适当测试图形、用于测试能够建模物理缺陷的逻辑故障的关键流程。

对于被测电路中的目标故障，自动测试图形生成的目标是：为目标故障生成测试图形（即找出能够区分正常机器与故障机器，并使该故障可检测的测试向量），或者证明该故障不可检测（即不存在能够区分正常机器与故障机器的测试向量）。目前已有多种用于自动测试图形生成的结构型算法，例如 D-算法 [1] 及其改进变体 [2], [3]。

ATPG 中有两个核心问题。其一是在给定回溯限制的情况下，如何提高决策效率，尤其是考虑到当今复杂设计中存在大量难检测故障。解决这一问题主要有两种方法。一种是直接利用布尔可满足性（SAT）求解器 [4], [5]。与在电路网络上工作的结构化 ATPG 不同，基于 SAT 的 ATPG 利用符号计算技术，在合取范式（CNF）上实现高效的冲突驱动搜索。已有许多基于 SAT 的 ATPG 算法，例如 TG-Pro [6], TIGUAN [7], 和 PASSAT [8]。类似的基于 SAT 的技术也已用于插入测试点，以实现低捕获功耗测试，同时保持相同的故障覆盖率 [9]。

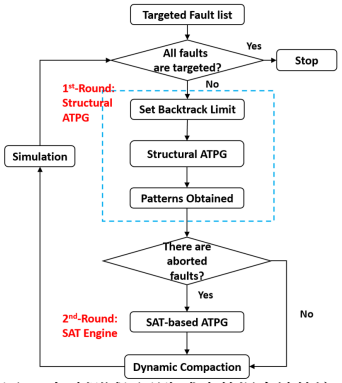

图 1. 自动测试图形生成中的混合计算流程，从结构型自动测试图形生成开始，以 SAT 结束。

如图 1 [4]所示，已提出一种由结构化 ATPG 和基于 SAT 的算法组成的混合计算流程。在给定的回溯限制下，首先采用结构化 ATPG 算法，其目标是相对容易检测的故障，这些故障可通过测试向量检测出来，或被证明不可检测。随后，SAT 针对被结构化 ATPG 中止的难检测故障。不同于结构化 ATPG，结构化 ATPG 直接在电路上执行，而基于 SAT 的算法依赖于由故障传播和激活的逻辑锥转换而来的合取范式。该转换是基于 SAT 的算法中的一个额外步骤。

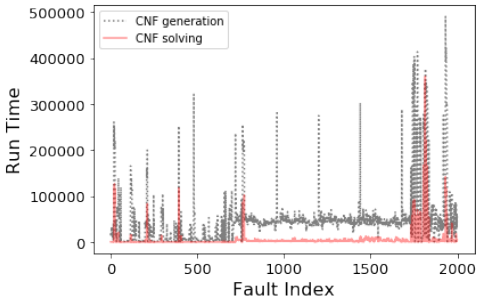

图2. 合取范式生成时间与求解时间的比较（单位：CPU 微秒）。横轴为故障索引，纵轴为相应的运行时间。

以电路 *Stuck* 4 为例（更多细节见第 IV 节），我们在图 2 中展示了 SAT 面临的一项重大挑战。该图比较了将电路转换为合取范式所需的时间与相应的求解时间。所有目标故障均为固定型故障，且基于 SAT 的框架采用 TG-Pro [6]。所选的 SAT 求解器为 SAT竞赛 2022年中的参考 SAT 求解器 Kissat [10]。

结果显示，转换过程所需的运行时间超过了求解本身。这表明，尽管 SAT 求解器取得了显著进展，并展现出解决自动测试向量生成问题的巨大潜力 [5]，但合取范式转换所需的额外开销限制了 SAT 在工业自动测试向量生成中的应用。为缓解这一问题，人们已开展了多项研究。有人提出了增量式 SAT 框架，旨在增量生成合取范式，并通过部分合取范式找到最终解或证明其不可满足，从而平均减少转换时间和求解时间 [4]。人们还采用预处理方法简化故障传播与激活的逻辑锥，通过求解等价替代问题，减少生成和求解时间 [11]。

然而，对大规模电路进行合取范式转换仍是一个主要瓶颈，这直接限制了 SAT 求解器的应用。因此，第二种方法是尝试在电路上利用 SAT启发式方法。有人提出了一种结合动态决策排序技术的冲突驱动递归学习方法，用于解决难以处理的故障 [12]。有人提出了一种冲突驱动蕴含方法，以提高证明效率并避免测试向量过度指定 [13]。此外，不可测试调试算法也已用于低测试覆盖率分析 [14]。然而，现代 SAT 求解器中构造学习到的冲突的方法（例如唯一蕴含点（UIP））尚未得到考虑。

另一个问题是，在设计早期阶段，ATPG 约束通常较为保守 [14]。这种保守性往往导致实际应用中的实现尚不够成熟。因此，在早期阶段，可测性设计工程师有一定的自由度，可以放宽或修改某些约束，从而有可能解决一些并非由电路结构导致的中止故障和不可测试故障。为解决这一问题，我们在运行 ATPG引擎后采用冲突诊断方法，以解决低测试覆盖率问题。以一个中止故障为例。我们认为，该故障中止是因为遇到的冲突超过了回溯限制。最后，对学习到的冲突进行统计分析，可以为可测性设计工程师提供有意义的建议，从而减少中止故障或不可测试故障的数量，并提高覆盖率。

受上述讨论启发，本文提出一种冲突驱动结构学习（CDSL）自动测试图形生成算法，旨在利用结构化 ATPG 与基于 SAT 的算法各自的优势。总而言之，我们的贡献包括：

(i)我们首先直接在电路上构建基于已学习冲突的约束，旨在利用优化过程数据剪枝搜索空间。根据与决策过程直接相关的蕴含图，我们构建了两类冲突约束，即基于决策变量的约束和基于唯一蕴含点的约束，从而避免后续迭代中的无意义搜索。

(ii)我们采用冲突驱动决策规则来提高决策准确性。 在累积学习到的冲突后，我们基于这些冲突构建新的蕴含和正当化方法。大量实验表明，冲突约束能够提升蕴含效率，并减少回溯和中止故障。

(iii)在低覆盖率调试的情况下，我们进一步根据学习到的冲突构建冲突诊断。在该方法中，我们利用学习到的冲突分析主输入约束方面的原因，并放宽或修改其中的某些约束，以进一步提高测试覆盖率。

本文其余部分安排如下。第二节介绍一些预备知识，第三节阐述我们提出的新型基于 SAT 的自动测试图形生成方法。第四节展示实验结果，并说明所提出框架在解的质量和运行时间方面的有效性。最后，我们在第五节总结本文工作。

II. 预备知识

A. SAT 中的冲突驱动子句学习（CDCL）

基于 SAT 的自动测试向量生成利用高效的 SAT求解器来解决 APTG 问题。它首先构建一个合取范式格式的 SAT 模型，该模型表示故障条件以及主输入端和主输出端之间的传播。一般而言，合取范式公式 ϕ 由子句的合取 ω组成，其中每个子句表示文字的析取。文字可以是变量 x_i 或其补。每个变量可被赋予逻辑值 0 或 1。任何一般的布尔问题都可以表示为合取范式公式模型。SAT 求解器要么找到使 ϕ 满足的赋值，要么证明不存在这样的赋值，即不可满足。现代 SAT 求解器中的一个关键启发式方法是冲突驱动子句学习（CDCL）算法 [5]。一般而言，CDCL 是一种分支定界（BB）搜索框架，其中每一步都会选择一个文字和一个命题值（0 或 1）用于分支。CDCL 的一个关键特征是在回溯搜索过程中从冲突中学习新子句。

B. 结构化 ATPG 算法

与基于 SAT 的算法不同，结构化 ATPG 算法直接在电路上执行。目前已有多种算法，例如 D 算法、PODEM 和 FAN。在实践中，D 算法会尝试将固定型故障值传播到主输出端（PO） [1]，该值表示为 D （表示固定为 0）或 D （表示固定为 1）。传统的 D 算法会生成决策结构，以评估电路中每个节点的值，从而获得测试向量。PODEM 和 FAN 是通过限制搜索空间并加快回溯而改进的算法变体；其中，PODEM 将搜索空间限制在主输入端（PI） [15]，，而 FAN 将搜索空间限制在头节点 [16]。

C. 结构化 ATPG 与 CDCL 的比较

CDCL 与结构化 ATPG 算法之间存在一定差异。首要差异必然源于分支规则。结构化 ATPG 算法由需求驱动 [1]，，这意味着决策顺序遵循故障传播和电路结构特征。与此不同，CDCL 的初始决策顺序遵循输入文字顺序，而该顺序是随机的；经过若干次回溯后，会根据文字在已学习的冲突约束中的出现频率对其进行调整。第二个差异在于发生冲突后的回溯规则。我们通过一个示例来讨论其他差异，如图 3 所示。所有决策变量（$x0$、 $x2$、 $x3$ 和 $x4$）都位于方框中，

而所有蕴含变量都位于椭圆框中。每个决策变量都根据决策顺序分配一个决策层级。箭头方向与蕴含方向一致。

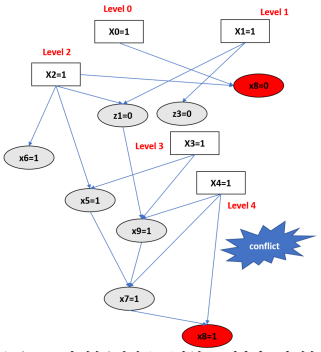

图 3. 决策过程示例。所有决策变量都用方框表示，蕴含用椭圆框表示。图中还标出了相应的决策层级。

图 3 显示，在第四个决策变量之后发生了冲突（即，$x8$ 不能同时为 0 和 1）。在结构化 ATPG 算法中，决策指针会回溯到上一个决策变量（即 $x3$），但不会分析冲突发生的原因。在给定的冲突驱动方法 [12]–[14], 中，会添加一个学习到的冲突约束 $x4 \neq 1$，以便在新的搜索规则下限制后续蕴含。显然，更好的搜寻结构化 ATPG 和 CDCL 两者的优势，即分支规则遵循结构化 ATPG 算法，旨在降低错误决策的代价；而一旦发生冲突，就应像 CDCL 一样考虑冲突原因，以避免再次沿着相同的错误搜索路径进行搜索。

III. 提出的 CDSL 算法

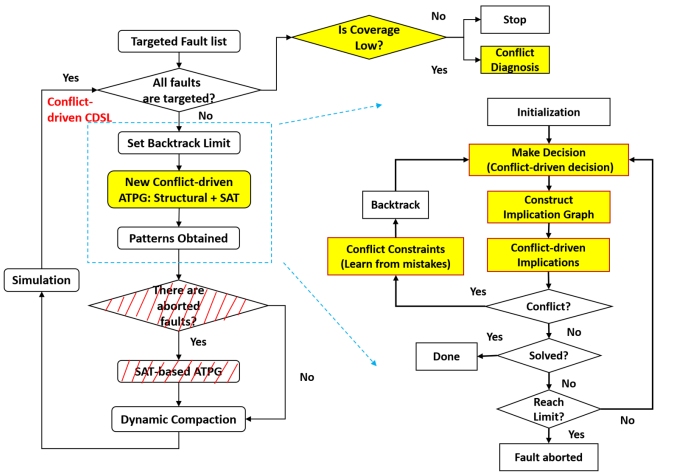

图 4. 新提出的 CDSL 算法。与传统的结构化 ATPG 算法不同，我们引入 SAT 启发式方法，例如已学习的冲突约束、冲突驱动蕴含和冲突驱动分支/决策，旨在根据优化过程中的数据剪枝搜索空间，并以更少的回溯找到解或证明不可满足。完成新的 ATPG计算后，我们提出在覆盖率较低时加入冲突诊断。

鉴于上述内容，我们提出了一种冲突驱动结构学习（CDSL）ATPG 算法，它结合了两种方法，如图 4 所示。与传统的结构化 ATPG 和基于SAT的ATPG算法相比，CDSL 算法具有两项优势：（1）它会累积回溯后的冲突约束，旨在避免重复作出错误决策，并以更少的回溯找到解。（2）它利用冲突驱动蕴含来剪枝搜索空间，并采用带评分启发式方法的冲突驱动分支规则，以提高决策准确性。

给定一个故障点，我们首先追踪电路，以获取与故障传播和激活相关的逻辑锥。决策规则从故障点开始，并遵循传统的结构化 ATPG 算法，直到出现冲突。在此过程中，可以使用所有结构化 ATPG 算法，例如 D 算法、PODEM 和 FAN。

A. 蕴含图

首先，我们根据决策过程构建一个蕴含图：

(1) 我们构建一个有向无环图，其中每个顶点表示一个变量的赋值，而连接到某个顶点的每条关联边都表示导致该赋值的原因。如果一个蕴含是通过其他蕴含推断出来的，那么不同蕴含之间也存在一条边。因此，与在传播过程中被强制赋值的蕴含变量不同，决策变量没有关联边。

(2) 每个决策变量都根据相关的决策顺序分配一个决策层级，而其相关的蕴含具有相同的决策层级。

请注意，CDSL 的蕴含图中的每个变量都表示一个逻辑门。一旦发生冲突，所提出的 CDSL 算法便会追踪蕴含图，找出所有导致冲突的历史赋值，并构造学习到的冲突约束。

B. 已学习的冲突约束

以图 3 为例，其中冲突由 x_8 引发。在所提出的 CDSL 算法中，我们构造两类已学习的冲突约束。

(1) 基于决策变量的冲突。其基本原理是，当前冲突至少由所有历史决策变量引起。如图 3 所示，冲突发生前有四个决策变量，即 $x_0 = 1$、$x_1 = 1x_2 = 1$、$x_3 = 1$ 和 $x_4 = 1$，因此我们可以添加一个由这些决策变量构造的已学习冲突约束 $x_0 + x_1 + x_2 + x_3 + x_4$。这表示在后续决策过程中，即使这四个变量都可作为决策变量，也不能重复相同的赋值；换言之，当发现其中三个变量重复了历史赋值时，第四个变量必须采用相反的赋值。

(2) 基于唯一蕴含点（UIP）的冲突。UIP 是当前决策层级中的一个特殊节点，从决策变量到冲突节点的任何路径都必须经过该节点 [17]。如图 3 所示，冲突发生在节点 x_8，其决策层级为 4。基于 UIP 的学习冲突约束的推理过程如下：

(i) 首先，我们找出冲突节点的直接原因。图 3 显示，x_8的一个直接原因是 x_4 和 x_7，另一个 x_8的直接原因是 x_0 和 x_2。其中，x_0、x_2和 x_4 均为决策变量，其决策层级分别为 0、2 和 4。x_7 是由 x_4、x_5和 x_9蕴含得到的。因此，直接学习到的冲突约束可表示为 $x_0 + x_2 + x_4 + x_7$。

(ii) 检查决策层级，并确定是否应将某些原因节点替换为相应的父节点。评估规则是：在最终学习到的冲突约束中，只有一个变量的决策层级与冲突节点相同，该变量即为唯一蕴含点（UIP）。

(ii-a) 考虑 $x_0 + x_2 + x_4 + x_7$，由于 x_7、x_9 和 x_4 均处于决策层级 4，且 x_4 是一个决策变量，因此我们利用 x_7 的父节点（即 x_4、x_5 和 x_9）将其替换。去重后，学习到的冲突约束更新为 $x_0 + x_2 + x_4 + x_5 + x_9$，其中 x_5 和 x_9 的决策层级分别为 3 和 4。

(ii-b) 由于 x_9 和 x_4 处于相同的决策层级，因此我们利用 x_9 的父节点（即 z_1、x_3 和 x_4）将其替换，随后学习到的冲突更新为 $x_0 + x_2 + x_4 + x_5 + z_1 + x_3$。

最后，我们可以得到基于 UIP 的学习到的冲突约束 $x_0 + x_2 + x_4 + x_5 + z_1 + x_3$。考虑到只有变量 x_4 的决策层级与冲突节点相同，因此 x_4 充当 UIP 节点。请注意，我们仅展示不同变量之间学习到的关系，不包括逻辑值。在累积了不同的学习到的冲突约束后，所提出的 CDSL 算法将在以下三个方面利用这些约束：

C. 冲突驱动蕴含

所有已学习的冲突约束都会应用于蕴含阶段，以避免重复错误的搜索路径。以图 3 中基于 UIP 的学习冲突约束 $x_0 + x_2 + x_4 + x_5 + z_1 + x_3$ 为例，如果我们发现五个相关变量（即 x_0、x_2、x_4、x_5 和 z_1）的赋值与历史赋值相同，那么第六个变量必须赋予相反的值。为避免累积过多已学习的冲突约束而产生额外的计算开销，我们还在蕴含阶段加入了一条遗忘规则：如果某个学习到的冲突约束在最近的 N 个循环中未被使用，就认为该约束已不再相关，并在接下来的循环中将其删除。因此，N 是一个超参数。

D. 冲突驱动分支启发式

学习到的冲突约束也可以通过变量状态独立衰减和（VSIDS）启发式方法应用，旨在提高后续决策阶段的决策准确性。VSIDS 策略分为三个步骤：

a) 我们首先为每个变量分配一个浮点分数。发生冲突时，某些变量的活跃度会增加 1。通常，初始分数设为 0。

b) 每次冲突后，变量活跃度会定期衰减，旨在权衡历史决策与后续决策。该衰减因子设为 $[0, 1]$。

c) 为了平衡 VSIDS 与结构策略，我们会在分支过程中检查每个变量的评分。以给定概率选择评分最高的变量。

此外，与需要回溯到最后一个决策变量的结构化 ATPG 算法不同，我们在所提出的 CDSL 算法中采用**非时序回溯**规则。该规则符合基于 UIP 的冲突约束，回溯点是除 UIP 节点外决策层级最高的变量。以图 3 为例，在累积基于决策变量和基于 UIP 的冲突约束后，x_0、x_5、x_3 和 x_4 的评分均高于其他变量；一旦发生冲突，便选择 x_3作为回溯点。

E. 低覆盖率调试的冲突诊断

除了蕴含和分支之外，我们还探索采用冲突诊断来解决设计初期测试覆盖率较低的问题：

(i) 根据各逻辑门出现在已学习的冲突约束中的频率，计算其评分。

(ii) 按评分排名选出前 k 个逻辑门。随后沿电路追踪，找出相关的外部约束。通常，这些约束施加在主输入或决策层级 0 的扇入门上。

在冲突诊断中，我们选择放宽或修改已识别的外部自动测试向量生成约束，从而有机会使中止故障或不可测试故障变得可测试。

IV. 实验结果

A. 实验设置

本节旨在从以下三个方面评估所提出的 CDSL 算法：

研究问题1：与传统的基于 SAT 的算法相比，它是否具有性能优势？**研究问题2**：与结构化算法相比，它是否有助于提高测试覆盖率？**研究问题3**：能否利用冲突诊断来调试中止故障或不可测试故障？

下文中，CDSL 框架基于结构化 D 算法实现，并从两个角度评估其性能：一是在设定的中止上限下，中止故障（未观察到的故障，简称 UO）的数量；二是故障覆盖率，即故障覆盖率$\frac{N_{Testable}}{N_{Total}}$，其中 N_{Total} 和 $N_{Testable}$ 分别表示故障总数和可测试故障数。所有实验均针对工业电路开展，其设计见表 I。

表 I 设计特性

电路	故障类型	门数量	状态数量	电路	故障类型	门数量	状态数量	
卡住 1	固定型故障	246078	14979	Tran 1	转换	139871	9644	
卡住 2	固定型故障	246078	14979	Tran 2	转换	785559	26288	
卡住 3	固定型故障	221004	18190	Tran 3	转换	785559	383963	
Stuck 4	固定型故障	78600	12047	Tran 4	转换	785559	357483	
卡住 5	固定型故障	221004	18190	Tran 5	转换	221004	357483	
卡住 6	固定型故障	206221	15772	Tran 6	转换	221004	331291	
卡住 7	固定型故障	56586	8194	Tran7	转换	221004	374009	
卡住 8	固定型故障	221004	357483	Tran 8	转换	206221	331291	
卡住 9	固定型故障	246078	331291	Tran 9	转换	206221	331291	
卡住 10	固定型故障	785559	26288	Tran 10	转换	221004	331291	

B. 运行时间评估

为回答 **研究问题1**，我们选择固定型故障，将所提出的 CDSL 与基于 SAT 的方法进行比较，如表 II 所示。第一列为电路名称。第二列和第三列分别列出所提出的 CDSL 算法导致的中止故障数量及相应的运行时间（CPU秒）。在此，中止上限设为 100。接下来，从左到右依次是四种用于评估 CDSL 算法的不同基线方法：

i) 基础的基于 SAT 的框架 TG-Pro [6]。它也是最新的开源框架。SAT 求解器选用 Kissat2022 [10]。

ii) 基础的 D 算法。它也是所提出的 CDSL 算法的一个模块。

iii) 一种带有预处理流程的增量型基于 SAT 的自动测试向量生成方法 [4]。

iv) 一种带有故障分析模块的基于 SAT 的自动测试向量生成方法 [18]，该模块是经过训练的神经网络，可预测故障分类，以便选择合适的算法。

结果表明，与传统的基于 SAT 的自动测试向量生成方法和结构化 D 算法相比，所提出的 CDSL 算法平均可分别减少 25.6% 和 49.88% 的中止故障，同时分别缩短 94.51% 和 25.88% 的运行时间。尽管这两种新变体，即带预处理的基于 SAT 的自动测试向量生成方法，以及采用已训练网络进行基于网络的故障分析的基于 SAT 的自动测试向量生成方法，能够减少中止故障并缩短运行时间，但所提出的 CDSL 还可分别将 UO 减少 45.23% 和 12.35%，相应的运行时间则可分别缩短 58.79% 和 93.09%。

值得一提的是，在回溯限制相同的情况下，传统的结构型自动测试图形生成方法和所提出的 CDSL 算法都比基于 SAT 的方法产生更少的中止故障。这是因为 SAT启发式方法（例如分支、重启和局部搜索）完全依赖于根据累积冲突计算的评分。这表明，有限的冲突约束可能会影响启发式方法的性能。

表 II CDSL 在 UO 上的性能和运行时间

电路	CDSL		TG-Pro		结构型		增量型		神经网络	
	UO	时间	UO	time	UO	time	UO	time	UO	time
卡住 1	147	229	174	10952	226	814	162	1528	162	9125
卡住 2	352	167	559	1722	793	128	638	218	475	1522
卡住 3	253	33	195	780	271	58	139	678	175	672
Stuck 4	1	53	7	1103	8	101	12	206	7	856
卡住 5	144	18	119	393	158	36	105	79	110	326
卡住 6	1343	365	1318	5165	1949	1307	2125	806	986	4238
卡住 7	236	97	485	1389	453	92	383	234	429	1109
卡住 8	601	550	518	10543	664	498	836	631	492	7692
卡住 9	514	75	987	977	1303	812	1189	235	836	901
卡住 10	545	878	1197	11931	1028	984	1963	1368	975	9312
平均值	**414**	**247**	556	4496	825	333	755	598	465	3569
改进 /		/	25.6%	94.51%	49.88%	25.88%	45.23%	58.79%	12.35%	93.09%

C. 覆盖率评估

为了进一步比较所提出的 CDSL 与结构化算法，我们针对跳变故障构建了一个两阶段自动测试模式生成框架。(i) 在第一阶段，我们设置较小的回溯限制，并关闭冲突驱动模块。我们的目标是以较小的中止上限处理易于检测的故障（中止上限设为 20）。(ii) 在第二阶段，我们设置较大的中止上限，并由所提出的 CDSL 算法处理中止故障（中止上限设为 100）。以下实验采用两种基线方法：(1) 第一种基线方法是单阶段传统 D 算法。(2) 第二种也是两阶段算法，但两个阶段均关闭冲突驱动模块。结果如表 III 所示。

结果发现，单阶段传统 D 算法平均产生 8702 个中止故障，故障覆盖率为 92.95%。然而，当相同的 D 算法采用两阶段设置时，中止故障可减少至 5975 个，故障覆盖率可达到 95.21%。此外，当所提出的 CDSL 采用两阶段设置时，中止故障可减少至 4667 个，故障覆盖率可提高至 96.14%。换言之，与 D 算法相比，中止故障可减少 46.37%，故障覆盖率可提高 3.19%；与不含冲突驱动模块的两阶段算法相比，中止故障可减少 21.89%，故障覆盖率可提高 0.93%。

表 III 两阶段框架下的评估

电路	单阶段		无冲突		所提模型	
	UO	覆盖率	UO	覆盖率	UO	覆盖率
Tran 1	505	95.57%	402	96.785%	353	97.149%
Tran 2	32319	98.71%	22710	99.109%	17154	99.325%
Tran 3	105	97.86%	119	98.867%	98	99.029%
Tran 4	604	97.59%	320	98.611%	214	98.928%
Tran 5	5414	91.71%	3769	94.678%	2943	95.795%
Tran 6	13211	90.55%	9110	93.548%	7339	94.777%
Tran 7	14037	90.15%	9462	93.383%	7615	94.634%
Tran 8	13436	90.50%	9152	93.603%	7364	94.819%
Tran 9	1641	88.34%	671	91.342%	526	93.011%
Tran 10	5757	88.53%	4043	92.25%	3067	93.97%
平均值	8702	92.95%	5975	95.21%	**4667**	**96.14%**
改进 46.37%	3.19%		21.89%	0.93%	/	/

D. 冲突诊断评估

最后，我们评估了低覆盖率分析中的冲突诊断。如第 III-E节所述，根据累积的学习到的冲突，我们首先标记排名前 5 的逻辑门。沿着这些已标记的逻辑门追踪电路后，可以找到与冲突相关的 PI节点，并将对应的逻辑值标记为 α_N （假设有 N 个相关 PI节点）。如果找到的 PI节点上存在约束，我们就放宽这些约束。否则，如果某个找到的 PI节点上没有任何约束，我们倾向于对该节点添加约束，并将逻辑值设为与 α 相反的值。最后，我们再次调用 ATPG引擎生成测试向量或证明其不可测试性。结果见表 IV。结果表明，经过冲突诊断后，中止故障平均减少了 8.89%，而故障覆盖率提高了 0.271%。

V. 结论

为解决基于 SAT 的框架带来的效率问题，同时利用现代 SAT 求解器的高效启发式方法，本文提出了冲突驱动结构学习（CDSL）自动测试图形生成算法，使结构化自动测试图形生成能够受益于 SAT 的

表 IV 冲突诊断评估

电路	UO	覆盖率	电路	UO		覆盖率
卡住 1	554	99.120%	Tran 1	306		97.337%
卡住 2	522	99.010%	Tran 2	14928	99.505%	
卡住 3	920	98.606%	Tran 3	82		99.210%
Stuck 4	8	99.803%	Tran 4	126		98.600%
卡住 5	852	97.679%	Tran 5	2812	96.004%	
卡住 6	35	99.786%	转换 6	7002	95.232%	
卡住 7	392	98.938%	Tran 7	7213	94.887%	
卡住 8	2356	96.022%	Tran 8	6579	94.872%	
卡住 9	5910	95.931%	Tran 9	442		93.859%
卡住 10	3827	99.873%	Tran 10	2913	93.953%	

冲突管理和冲突驱动分支等技术。所提出的 CDSL 算法由三部分组成：（1）在每次回溯之前构建已学习的冲突约束，旨在从错误中学习，并利用优化过程数据剪枝搜索空间。（2）将冲突驱动蕴含和正当化应用于决策与蕴含，以进一步提高求解效率和决策有效性。（3）尝试通过分析已学习的冲突进行冲突诊断，并放宽部分外部自动测试向量生成约束，以提高测试覆盖率和故障覆盖率。工业电路上的大量实验结果证明，所提出的 CDSL 自动测试图形生成算法在以下三个方面具有优势：（i）与传统的基于 SAT 的自动测试向量生成和结构化 D 算法相比，所提出的 CDSL 算法使中止故障的平均数量分别减少了 25.6% 和 49.88%，同时运行时间分别缩短了 94.51% 和 25.88%。（ii）在两阶段设置下，与 D 算法相比，中止故障减少了 46.37%，故障覆盖率提高了 3.19%；与不含冲突驱动模块的两阶段算法相比，中止故障减少了 21.89%，故障覆盖率提高了 0.93%。（iii）实验表明，冲突诊断平均使中止故障减少了 8.89%，同时使故障覆盖率提高了 0.271%。未来工作包括针对结构化自动测试向量生成启发式方法开发更多 SAT 启发式方法。

参考文献

[1] J. P. Roth，《自动机故障诊断：一种演算与方法》，*IBM 研究与开发杂志*，第10卷，第278–291页，1966年。[2] N. Wang、C. Wang、K.-H. Tsai、W.-T. Cheng、X. Lin、M. Kassab和I. Pomeranz，《Tea：一种用于具有时序例外的设计的测试生成算法》，亚洲测试研讨会，第19–195页，2019年。[3]M. Schulz、E. Trischler和T. Sarfert，《Socrates：一种高效的自动测试向量生成系统》，国际测试会议，第1016–1026页，1987年。[4]J. Huang、H. L. Zhen、N. Wang、M. Yuan、H. Mao、Y. Huang和J. Tao，《通过预处理和新的冲突管理启发式方法加速基于 SAT 的自动测试向量生成》，第*27*届亚洲及南太平洋设计自动化会议（*ASP-DAC*），第365–370页，2022年。

[5] B. Becker、R. Drechsler和M. Sauer，《基于 SAT 的自动测试向量生成的最新进展：非标准故障模型、多重约束与优化》，纳米时代集成系统设计与技术国际会议，第1–10页，2014年。[6]H. Chen和J. Marques-silva，《Tg-pro：一种基于 SAT 的自动测试向量生成系统说明》，可满足性、布尔建模与计算期刊，第8卷，第1-2期，第83–88页，2011年。[7] A. Czutro、I. Polian、M. Lewis、P. Engelke、S. M. Reddy和B. Becker，《Tiguan：利用可满足性分析的线程并行集成测试向量生成器》，国际超大规模集成电路设计会议，第227–232页，2009年。[8]S. Eggersgl¨uß、K. Schmitz、R. Krenz-B˚a˚ath和R. Drechsler，《基于优化的自动测试向量生成及其在高度压缩测试集中的应用》。*IEEE*集成电路与系统计算机辅助设计汇刊，第2104–2117页，2016年。[9]S. Eggersgl¨uß、S. Holst、D. Tille、K. Miyase和X. Wen，《面向基于区域的低捕获功耗紧凑型高速扫描测试的形式化测试点插入》。*IEEE*亚洲测试研讨会（*ATS*），第173–178页，2016年。[10] M. S. Cherif、D. Habet和C. Terrioux，《Kissat mab：结合 VSIDS 和 CHB 的上置信界策略》，*SAT*竞赛，2022年。[11] D. Tille、S. Eggersgluss和R. Drechsler，《基于 SAT 的自动测试向量生成的增量求解技术》，*IEEE*集成电路与系统计算机辅助设计汇刊，第29卷，第7期，第1125–1130页，2010年。[12] C. Wang、S. M. Reddy、I. Pomeranz、X. Lin和J. Ra-jski，《用于改进确定性测试向量生成的冲突驱动技术》。*IEEE/ACM*计算机辅助设计国际会议，第87–93页，2002年。[13] S. Bommu、K. Chandrasekar、R. Kundu和S. Sengupta，《Concat：工业设计中自动测试向量生成的冲突驱动学习》。*IEEE*国际测试会议（*ITC*），第1–10页，2008年。[14]C. Kameshwar、S. Bommu和S. Sengupta，《利用自动测试向量生成中的动态不可测试性调试进行低覆盖率分析》。*IEEE*超大规模集成电路测试研讨会（*VTS*），第291–296页，2011年。[15] P. Goel，《一种为组合逻辑电路生成测试的隐式枚举算法》，*IEEE*计算机汇刊，第C-30卷，第215–222页，1981年。[16] K. T.和M. R. Mercer，《一种用于自动测试向量生成的拓扑搜索算法》。第*24*届*ACM/IEEE*设计自动化会议，第502–508页，1987年。[17]M.-S. Joao、I. Lynce和S. Malik，《冲突驱动子句学习 SAT 求解器。可满足性手册》。*IOS Press*，第133–182页，2021年。[18]J. Huang、H. L. Zhen、N. Wang、M. Yuan、H. Mao和Y. Huang，《面向基于 SAT 的自动测试向量生成的神经故障分析》。*IEEE*国际测试会议（*ITC*），第36–45页，2022年。