arXiv:2304.04292v1 [cs.LO] 9 Apr 2023

使用高斯-约旦消元为 CDCL 求解器生成证明

Mate Soos[1] 和 Randal E. Bryant[2] *

[1] 新加坡国立大学 soos.mate@gmail.com[2] 卡内基梅隆大学计算机科学系，美国宾夕法尼亚州匹兹堡 Randy.Bryant@cs.cmu.edu

摘要

基于冲突驱动子句学习（CDCL）框架的传统布尔可满足性（SAT）求解器，在处理包含大量奇偶约束的公式时表现不佳。CryptoMiniSat 求解器通过将高斯-约旦消元融入 CDCL，大幅提升了处理这类公式的性能。将 tbuddy 证明生成 BDD 库集成到 CryptoMiniSat 中，使其能够在使用高斯-约旦消元时生成不可满足性证明。这些证明与标准的子句证明框架兼容。

1 引言

考虑定义在一组变量 $X = \{x_1, x_2, \ldots, x_n\}$ 上的布尔公式。 k 路奇偶约束，也称为异或约束，是如下形式的方程：

$$x_{i_1} \oplus x_{i_2} \oplus \cdots \oplus x_{i_k} \quad = \quad p \tag{1}$$

其中，相位 p 可以为 1（奇校验）或 0（偶校验）。

尽管基于冲突驱动子句学习（CDCL）的布尔可满足性（SAT）求解器多年来不断取得改进，但当公式包含大量奇偶约束时，它们的表现仍然不佳。

例如，Urquhart 构造了一族不可满足公式，其完全由奇偶约束组成，其中变量数量和子句数量均随规模参数 m呈二次增长，但任何不可满足性证明的归结证明规模都必须随 m [27] 呈指数增长。该基准测试的最小实例为 $m = 3$，包含 153 个变量和 408 个子句，编码了 102 个三元奇偶约束。 CryptoMiniSat 是一款先进的 CDCL 求解器 [3], ，即使禁用证明生成，在该公式上运行 16 小时后仍未能终止。另一方面，将式 1 形式的约束视为模 2 整数上的线性方程，对不可满足的奇偶约束集合应用高斯消元，可在多项式时间内得到无解方程 $0 = 1$ 。

SeveralCDCL 求解器已通过增强约束求解器来实现对奇偶约束进行高斯-约旦消元 [12,13,17,24]。这些求解器将传统的子句推理与奇偶推理相结合，以提升其在可满足和不可满足公式上的性能。它们首先检测输入公式中编码的奇偶约束，并将这些奇偶约束委托给奇偶校验推理组件，同时保留

*由美国国家科学基金会通过资助项目 CCF-2108521 提供支持

用于子句推理组件的其余子句。在执行期间，两个组件分别对公式中各自的部分进行推理，并通过单元传播和冲突检测进行协调。这一机制有时被称为“CDCL(T)”，体现了其与 SMT 求解器处理多个理论的相似性 [9]。这种方法可能极其有效。例如，CryptoMiniSat 5.8.0 版本在运行 Urquhart 公式时，$m = 3$ 仅用一秒就能检测出该公式不可满足。即使扩展到 $m = 40$，一个大 200多倍的公式，其中包含33,120个变量和 88,320 个子句，该程序也能在不到五分钟内检测出该公式不可满足。

1.1 SAT 求解器的不可满足性证明

新一代 CDCL SAT 求解器在遇到不可满足公式时，可以生成不可满足性证明。这种证明提供了可独立检查的确认，表明该公式确实不可满足。自 2016 年起，年度 SAT 竞赛主赛道的参赛者只有在满足以下条件时，其不可满足结果才能获得计分：1）生成 DRAT 证明 [28], ；2）该证明通过标准证明检查器的成功检查。不可满足性证明还确保了采用 SAT 求解的形式化验证工具和数学证明的完整性。当结果旨在为航空航天、交通运输和网络安全等高风险环境提供保障时，这一点尤为重要。

迄今为止，在采用奇偶推理的同时生成子句证明一直是一项重大挑战。标准的子句证明框架（例如 DRAT）与 CDCL 求解器十分匹配，但在采用高斯消元或高斯-约旦消元的同时生成 DRAT 证明一直进展有限。因此，近年来 SAT 竞赛主赛道的参赛者并未使用奇偶推理。[1]

由于在使用 Gauss-Jordan 消元时无法生成子句证明，大多数当前的 SAT 求解器在被要求生成证明时会禁用奇偶推理，转而完全依赖 CDCL。在这种模式下，对于包含奇偶约束的公式（包括 Urquhart 公式），它们的表现并不优于 Kissat。

1.2 相关工作

有些人提出采用其他证明框架，以支持奇偶约束求解器进行证明生成。例如，VeriPB 证明检查器会验证以伪布尔约束逻辑表示的证明[11]。以该框架为目标，Gocht 和 Nördstrom 得以将高斯-约旦消元求解器集成到MiniSat 求解器中，并使其生成不可满足性证明 [12, 23]。类似地，Barnett 和Biere提出了一个基于二叉决策图（BDD）的证明框架，其中每个证明步骤都保证能够以多项式复杂度进行检查 [2]。他们的框架也可以支持从奇偶约束生成证明。要让 SAT社区采用新的证明框架，就需要建立并记录一套新的文件格式，并创建一组证明检查器，最好其中还包括经过形式化验证的证明检查器。我们的方法具有以下优势：它建立在久经验证的 DRAT 框架之上。

在之前的工作中，我们在推理 伪布尔 公式时，提出了一种从根本上不同的方法来生成不可满足性证明，其中奇偶约束是一个特例[5]。我们通过将高斯消元与基于二叉决策图的证明生成器相结合来实现这一点。二叉决策图对于推理奇偶约束尤其有效——二叉决策图表示

[1] “nolimits”赛道允许求解器声明某个公式不可满足而无需提供证明，因此采用高斯-约旦消元的求解器继续参加该赛道。

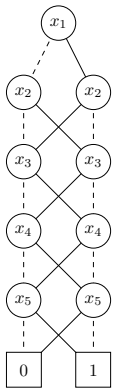

图 1：5 个变量的奇偶约束的二叉决策图表示。无论二叉决策图变量采用何种排序，该表示的规模都与变量数量呈线性关系。

的 k元约束仅有 $2k-1$ 个非终端节点，无论二叉决策图变量排序如何。例如，图1 展示了奇偶校验约束的二叉决策图表示

$$x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \quad = \quad 1$$

证明生成器利用了在 DRAT 证明中引入扩展变量的能力，其风格类似于扩展归结 [16, 26]。为生成的每个二叉决策图节点添加一个扩展变量，用于描述该节点与其对应变量及子节点之间的逻辑关系 [6,15,22]。二叉决策图操作的每个递归步骤都由一小段 DRAT 证明步骤加以证明。这样，当高斯消元求解器检测到奇偶约束不可行时，整个证明便由一系列保持蕴含关系的子句组成，并以空子句结束。另一方面，对于许多 CDCL 求解器能够轻松处理的公式，我们基于二叉决策图的求解器表现不佳；还有第三类问题，需要同时进行子句推理和奇偶推理才能判定公式不可满足。将二叉决策图的证明生成能力整合到 CDCL(T) 求解器中，可以扩大其能够生成不可满足性证明的公式范围。

Philip 和 Rebola-Pardo 提出了一种在处理奇偶约束时生成 DRAT 证明的方法 [21]。他们的方法维护奇偶约束的子句表示，并使用辅助变量，以确保 k 元约束的表示只需 $O(k)$ 个子句。约束求解器每执行一步，该方法都会生成一个证明，说明新生成约束的子句编码在逻辑上蕴含于各个参数约束的编码。新约束中的辅助变量会成为证明中的扩展变量。从操作方式上看，该方法类似于基于二叉决策图、能够生成证明的求解器：每生成一个二叉决策图节点，就会添加扩展变量。他们的工作专门针对奇偶约束，而我们的方法则可应用于其他类别的伪布尔约束 [5]。Gocht 和 Nördstrom 将该方法纳入了他们扩展版的 MiniSat [12]。尽管他们能够为纯 CDCL 求解器无法处理的基准电路生成证明，但他们发现检查时间的扩展性不太理想。我们推测，这是因为标准检查器 drat-trim [28] 在含有大量扩展变量的公式上的表现不佳；我们也遇到了这一缺陷。

1.3 贡献

本文介绍了我们近期将二叉决策图的证明生成功能集成到CryptoMiniSat的高斯-若尔当（G-J）求解器组件中的经验。CryptoMiniSat 是一款经过高度优化的 CDCL(T) 求解器。我们使用tbuddy [4], ——一个通过扩展buddy [19] BDD 软件包实现的证明生成型二叉决策图库。[2] Tbuddy 会生成证明步骤，以证明高斯-若尔当消元过程中推导出的单元传播和冲突是合理的。该功能通过松散集成的方式加入，只向CryptoMiniSat添加了一个新的数据结构。

一个更大的挑战是生成能够高效检查的证明。 Drat-trim是 DRAT 证明的标准检查器，[28], 无法处理 tbuddy生成的大型证明，尤其是其中大量使用扩展变量的情况。幸运的是，我们通过采用混合 FRAT 证明格式的检查器，实现了可接受的性能 [1], ；许多证明步骤（包括 tbuddy 生成的所有步骤）都附有提示，大大简化了检查器的运行。尽管 FRAT 证明采用的文件格式不同于标准 DRAT 格式，但相关检查器 frat-rs 能够生成与 [14] 相同的详细 LRAT 格式证明，drat-trim也是如此。这样，两种检查器都能利用各种可用的 LRAT 证明检查器，其中一些已经过形式化验证 [14, 25]。

本文介绍了将Tbuddy 集成到CryptoMiniSat 中，并使用两类基准问题进行了实验评估。

2 求解器/证明器集成

对于具有方程 P1 所示形式的奇偶约束，令Var(P) $=\{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}$由该约束中的变量组成，令Phase($P$) $= p$ 表示其相位。约束P_a 与 P_b 之和也是一个奇偶约束 P_c，记作 $P_c = P_a \oplus P_b$，其中 Var(P_c) $=$ Var(P_a)$\triangle$Var(P_b)（即它们的对称差），且 Phase(P_c) $=$ Phase(P_a) $\oplus$Phase(P_b)。

令 ϕ 表示输入子句集合， P_1、 P_2、.....、 P_m 表示嵌入在输入公式中的奇偶约束集合。对于 $S \subseteq \{1, 2, \ldots, m\}$，令 $\Sigma(S)$ 表示将所有满足 $i \in S$ 的约束 P_i 相加后得到的奇偶约束。文字 ℓ 是一个变量或其补，其相位在前一种情况下为 1，在后一种情况下为 0。

集成到 CDCL框架中时，高斯-约旦消元提供了一种系统地求和奇偶约束的方法，从而检测出由约束 [13, 17]所蕴含的冲突和单元传播。假设搜索过程中的某个时刻，其状态由一个迹表示，该迹是文字 $\ell_1 \wedge \ell_2 \wedge \cdots \wedge \ell_k$的合取。当存在某个和 $\Sigma(S)$，满足以下条件时，就会产生冲突：1) $\Sigma(S)$中的变量是迹中出现的变量的子集；2) $\Sigma(S)$ 的相位与对迹中相应文字的相位求和所得的结果不匹配。当 $\Sigma(S)$中除一个变量外的所有变量都出现在迹中，且所蕴含的文字是该变量或其补文字，并且这些文字的相位之和与约束的相位一致时，就会产生单元传播。这两种推理都可以用一个原因子句 C 来刻画。对于冲突， C 是冲突子句，由迹中某些文字的补文字组成。换言之，要满足 ϕ，就必须违背迹中的至少一个赋值。对于单元传播， C 由某些迹文字的补文字以及通过单元传播推断出的文字组成。换言之，任何与迹相容的、满足 ϕ 的赋值也必须满足所蕴含的文字。

[2] Tbuddy 可在 https://github.com/rebryant/tbuddy-artifact。

为 CryptoMiniSat 添加奇偶约束的证明生成需要三项新能力：**1)** 生成证明步骤，以证明初始奇偶约束已编码在输入公式中；**2)** 跟踪产生每个原因子句 C的奇偶约束子集 S，以及 **3)** 根据这些约束生成证明步骤，以证明原因子句 C。

前两项能力中的第一项和第三项由 tbuddy 提供，第二项则需要为 CryptoMiniSat 添加跟踪功能。

2.1 可信二叉决策图（TBDD）

该TBUDDY 软件包支持将可信二叉决策图（TBDD）作为其核心数据结构。可信二叉决策图表示一个布尔函数，对于任何满足输入公式的赋值，该函数都保证返回 1。每个可信二叉决策图 $\dot{\mathbf{u}}$ 由以下部分表示：1）二叉决策图数据结构中的根节点$\mathbf{u}$；2）与二叉决策图节点 u $\mathbf{u}$ 相关联的扩展变量$\mathbf{u}$；以及 3）一系列证明步骤，这些步骤最终导出单元子句 $[u]$，表明任何满足该公式的赋值都必须同时将 u赋值为 1。执行一系列 TBDD 操作，最终生成可信二叉决策图$\mathbf{T}_0$，，其中$\mathbf{T}_0$ 是表示常量0的终端二叉决策图节点，其效果是生成一系列最终导出空子句的证明步骤。TBUDDY 支持对 TBDD 执行若干关键操作[4]:

合取的参数为 TBDDs $\dot{\mathbf{u}}$和 $\dot{\mathbf{v}}$。它生成一个TBDD $\dot{\mathbf{w}}$，表示由二叉决策图 $\mathbf{u}$ 和 $\mathbf{v}$ 表示的函数的合- 取。

upgrade 的参数是可信二叉决策图 $\dot{\mathbf{u}}$ 和普通二叉决策图$\mathbf{v}$。它根据 $u \to v$ 的证明将 $\mathbf{v}$ 升级为 TBDD $\dot{\mathbf{v}}$。

子句证明 的参数是受信任的二叉决策图 $\dot{\mathbf{u}}$和子句 C。它会生成一个或多个证明步骤，最终得到一个基于蕴含 $u \to C$、包含子句 C 的步骤。

2.2 论证初始奇偶约束的合理性

CryptoMiniSat 使用启发式方法检测以子句形式编码的奇偶约束。当它检测到一个包含 P_i 个变量的输入奇偶约束 k 时，它会让Tbuddy 生成该约束的 TBDD表示 $\dot{\mathbf{u}}_i$。为此，Tbuddy 只需将编码 P_i的 2^{k-1} 个子句添加到证明中。根据逆向单元传播（RUP），这些子句中的每一个都应当由输入子句的某个子集推出，因此证明检查器可以将必要的提示添加到证明中。基于这些提示，Tbuddy使用其合取操作构造一个 TBDD $\dot{\mathbf{v}}$，表示所添加子句的合取。它还直接生成 P_i的二叉决策图表示$\mathbf{u}_i$，其形式如图 1所示，但使用指定的变量集合和指定的相位。然后，它使用 $\dot{\mathbf{v}}$ 将 $\mathbf{u}_i$ 升级为 TBDD $\dot{\mathbf{u}}_i$。在大多数情况下，$\mathbf{u}$ 和 $\mathbf{v}$ 是相同的，因此该蕴含是重言式。

2.3 TrackingConstraint Origins

CryptoMiniSat 中 G-J 消元的标准实现CRYPTOMINISAT [23] 不必跟踪哪些原始奇偶约束导致了冲突或传播。相反，它可以直接根据奇偶矩阵推导出原因子句；奇偶矩阵是在高斯-约旦消元过程中维护的奇偶约束表示。为了加入跟踪功能，修改后的代码会维护一个二进制影子矩阵 M，其维度为 $n \times n$，对应含有 n 行的奇偶矩阵。启动时，M 会被初始化为单位矩阵。顾名思义，影子矩阵会跟踪奇偶矩阵上的操作。当奇偶矩阵中的两行互换时，影子矩阵

也会交换相应的两行；当两行相加时，影子矩阵也会将对应的两行相加。因此，在任何时候，若构造当前约束 i时，初始约束 j 被加了奇数次，则 M 的第 i, j 项设为 1。（请注意，将一个约束相加偶数次实际上会将其抵消。）借助影子矩阵，程序可以在奇偶矩阵中的某一行导致冲突或单元传播时，确定该行的来源。影子矩阵可以使用与奇偶矩阵相同的位向量表示法进行存储和操作。

示例。 设初始奇偶约束为 $\{P_1 := x_1 \oplus x_2 = 1, P_2 := x_1 \oplus x_3 = 0, P_3 := x_1 \oplus x_2 \oplus x_3 = 1\}$。那么，初始矩阵 M_0 及其对应的影子矩阵 S_0 如下所示。请注意， M_0 的最后一列包含相位值。由于 M_0 有 3 行，S_0 是 3×3 单位矩阵：

$$M_0 = \left[\begin{array}{ccc|c} x_1 & x_2 & x_3 & \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{array}\right] \qquad S_0 = \begin{bmatrix} P_1 & P_2 & P_3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

高斯-若尔当消元算法现在取出 M_0的第 1 行，并将其分别加到 M_0 的第 2 行和第 3 行，以消去 M_0 第一列中除第一行外的非零值。将所得矩阵记为 M_1 和 S_1:

$$M_1 = \left[\begin{array}{ccc|c} x_1 & x_2 & x_3 & \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{array}\right] \qquad S_1 = \begin{bmatrix} P_1 & P_2 & P_3 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

请注意， S_1的第 2 行现在是 110，也就是说，可以通过 $P_1 \oplus P_2$重建 M_1的第 1 行。类似地， S_1 的第 3 行是 101，也就是说，可以通过 $P_1 \oplus P_3$重建 M_1的第 3 行。这样，通过镜像执行行交换和行相加操作，始终可以借助 S_n 读出需要将哪些原始奇偶约束相加，才能得到 M_n中对应的约束。

2.4 为原因子句提供依据

每个原因子句 C 都必须依据初始奇偶约束的一个子集 S 加以证明。这需要先为和约束 $\Sigma(S)$ 构造 TBDD 表示 $\dot{\mathsf{v}}$，然后使用$\dot{\mathsf{v}}$ 为 C提供依据。给定分别表示奇偶约束 P_a 和 P_b的 TBDD $\dot{\mathsf{u}}_a$ 和 $\dot{\mathsf{u}}_b$，Tbuddy 可以生成奇偶约束 $P_c = P_a \oplus P_b$的 TBDD 表示 $\dot{\mathsf{u}}_c$。具体步骤如下：首先，将 $\dot{\mathsf{u}}_a$ 和 $\dot{\mathsf{u}}_b$ 合取，得到 TBDD $\dot{\mathsf{w}}$。然后，为 P_c生成 BDD 表示 u_c，其形式与图 1 所示相同。最后，通过证明蕴含关系 $w \to u_c$，将 u_c升级为 TBDD $\dot{\mathsf{u}}_c$。

约束 P_a 和 P_b 的合取的 BDD 表示 w 的大小可能达到 $\Theta(k_a \cdot k_b)$，其中 k_a 和 k_b 分别是 P_a 和 P_b中的非零系数个数。另一方面，它们的和的 BDD 表示 u_c 的大小为 $O(k_a + k_b)$。由于在多次消元步骤中，规模会不断增长，因此减少 BDD 运算（进而减少证明步骤）所带来的收益可能相当可观。

约束 S 之和的 TBDD 表示通过逐对求和生成。由于包含 k 个变量的约束的 BDD 表示有 $2k - 1$ 个节点，因此对约束求和时，必须尽量保持中间结果的稀疏性（即尽量减少约束中的变量数）。Tbuddy 通过贪心选择策略实现这一点：每一步都选择使以下值最小的约束对 P_a 和 P_b：

$\mathsf{Var}(P_a) \triangle \mathsf{Var}(P_b)$ 的大小；若出现并列，则随机打破平局。使用优先队列和增量更新即可高效完成此过程。

最后，子句论证操作会根据表示约束之和的 TBDD $\veebar$，将原因子句 C 添加到证明中。对于由奇偶约束产生的原因子句，单个 RUP 步骤即可满足要求。

一旦使用约束之和的 TBDD 表示来论证原因子句，就可以解除对它的引用，使 tbuddy 垃圾回收器能够回收节点并删除部分证明子句。仅当奇偶矩阵的行发生修改时，才会执行解除引用。因此，给定的行可以用于论证多个原因子句。

3 实验结果

我们针对多个对 CDCL 求解器而言尤其具有挑战性的不可满足公式族进行了实验评估。我们比较了 CRYP-TOMINISAT 的不同运行模式与 TBSAT的运行模式；后者是一个基于二叉决策图、构建于 TBUDDY之上的求解器。TBSAT 拥有自己的高斯-约旦消元求解器，并且可以将奇偶推理与 桶消元 [10, 15, 20], 相结合，以一种系统的方法，通过一系列合取和量化操作，将一组二叉决策图约简为两个可能终端节点之一 [4, 7]。

我们发现，标准 drat-trim 证明检查器在处理由 tbuddy 生成的证明时性能不佳，这是由于定义扩展变量时使用了大量 RAT 引理。因此，我们修改了 CryptoMiniSat，使其生成 FRAT 格式的证明 [1], ，其中 CryptoMiniSat 生成的证明子句有超过 50% 带有提示，而 tbuddy 生成的证明子句则全部带有提示。相关的 frat-rs 验证器可以利用这些提示，同时为其他子句生成提示。

3.1 Urquhart 公式

Urquhart 的公式 [27] 由基于无向图的奇偶约束集合构成，其方式与 Tseitin 的一种构造类似 [26]。每条边都关联一个变量，而公式针对每个节点及其相邻边包含一个偶校验或奇校验约束。每条边都会出现在两个约束中，因此，只要节点相位之和为奇数，该公式就不可满足。我们使用了 Li 编写的基准生成器 [18] ，它由图的规模 m 以及一个值 p 共同参数化，其中 $25 \leq p \leq 75$ 表示被赋予奇校验的节点所占的百分比。

如前所述，即使是这些公式中规模最小、含有 $m = 3$ 的实例，也超出了当今 CDCL 求解器的能力范围。另一方面，对这些约束应用高斯消元会得到不可行约束 $0 = 1$，因此仅使用奇偶约束推理就能轻松处理它们。我们还展示了，采用桶消元时，基于二叉决策图的求解器在这些公式上的规模增长是多项式级的 [4, 7]。

我们生成了困难度参数为 m 、范围从 3 到 15 且相位百分比参数为 p 、范围从26 到 70 的Urquhart 公式。测试在 AMDRyzen 95950X处理器上进行，每次运行的时间限制为 3600 秒。两种求解器的求解时间和验证时间如表 1 所示。最右列显示了 FRAT-RS 检查由 CRYPTOMINISAT 加上 TBUDDY生成的证明所需的时间。可以看出，Urquhart公式对两个求解器都不构成挑战，检查证明所需的时间与生成证明所需的时间相当。 TBSAT 在这些基准测试中相较于 CRYPTOMINISAT 的性能优势，可归因于其对奇偶约束采用了稀疏表示，以及该问题可以完全通过奇偶推理求解。

m	Tbsat lrat 已启用		lrat-check Tbsat 的		CMS Gauss-J 使用 FRAT	frat-rs Verify CMS 证明
	Gauss-J	Bucket-Elim	Gauss-J	Bucket-Elim		
3	0.00	0.00	0.69	0.69	0.03	0.06
4	0.01	0.01	0.69	0.69	0.10	0.17
5	0.03	0.03	0.69	0.69	0.24	0.40
6	0.05	0.05	0.69	0.69	0.46	0.84
7	0.07	0.07	0.69	0.69	0.87	1.66
8	0.10	0.10	0.69	0.69	1.44	2.96
9	0.13	0.13	0.69	0.69	2.08	4.34
10	0.16	0.17	0.69	0.69	3.33	7.30
11	0.20	0.21	0.69	0.69	5.37	12.26
12	0.25	0.26	0.69	0.69	5.00	11.73
13	0.31	0.30	0.69	0.69	7.25	16.61
14	0.37	0.36	0.69	0.69	10.36	23.96
15	0.43	0.43	0.69	0.69	14.10	34.47

表 1：使用 Tbsat 和 Crypto- MiniSat+tbuddy 求解并验证 Urquhart 公式的平均时间

3.2 使用噪声学习奇偶性

带噪学习奇偶性（LPN），也称为最小分歧奇偶问题，是 CDCL SAT 求解器面临的一个众所周知的难题。Crawford 为这个问题提供了若干公式实例，作为原始 SATLIB 基准套件的一部分 [8]。由于 Crawford 的生成器已无法使用，我们为这个问题编写了一个新的生成器，[3] 从可满足性求解的角度来看，这些公式将一组奇偶约束与一个基数约束结合起来。因此，高斯-约旦消元不会直接导出不可行约束，但在推理奇偶约束时仍能发挥作用，而子句推理部分则负责处理基数约束。

该问题由值 n、 m 和 k参数化。其中有 m 个奇偶约束，这些约束定义在一组 n 解变量 s_1、s_2、. . .、 s_n 以及一组 m 损坏变量 r_1、 r_2、. . .、 r_m 上。每个奇偶约束 P_i 取决于从解变量中随机选择的某个子集，以及损坏变量 r_i。生成实例时，对于由随机生成的比特序列 a_1、 a_2、. . .、 a_n 组成的目标解，当将解变量设为目标解并将其关联的损坏变量 r_i 设为 0 时， $n - k$ 个约束的子集将得到满足，而其中的 k 个约束要求 $r_i = 1$ ，目标解才能满足该约束。也就是说， k 个方程通过翻转其相位而被“损坏”。对损坏变量施加了至多k 个的约束，因此，只要损坏的约束数不超过 k ，目标解将满足该公式，其他解也可能满足该公式。标准的LPN 公式是可满足的，但通过施加至多 $k - 1$ 个损坏位可设为 1 的条件，该公式通常（但不一定）会变得不可满足。

对于从 20 到 35 的每个 n 值，我们生成了 20 个 LPN 公式实例，并设置$m = 2n$。每个公式中的每条约束都以 0.125 的概率随机损坏，因此每个公式中损坏约束的数量 k 有所不同，但

[3] 可访问https://github.com/rebryant/mdp-benchmark。

n	不可满足	CryptoMiniSat 不使用高斯-若尔当		CryptoMiniSat 使用高斯-若尔当	CryptoMiniSat 使用高斯-若尔当消元+tbuddy		frat-rs 平均值
		T.O.	PAR-2	无需验证	T.O.	PAR-2	（秒）
20	15	0	4.59	0.03	0	1.00	0.94
21	10	0	4.29	0.03	0	1.06	1.64
22	13	0	20.67	0.05	0	2.41	3.98
23	14	0	31.96	0.05	0	2.79	2.55
24	16	0	113.09	0.05	0	3.40	4.51
25	11	0	257.20	0.14	0	9.45	20.51
26	14	0	908.71	0.25	0	17.70	28.63
27	13	5	2401.46	0.31	0	22.91	42.79
28	8	5	2184.10	0.73	0	38.39	42.95
29	8	10	3800.17	1.27	0	70.64	159.45
30	14	12	4696.38	1.70	0	101.98	289.13
31	15	13	5120.66	3.07	0	194.07	726.44
32	12	15	5519.15	3.09	0	216.26	553.69
33	10	16	6167.31	6.71	0	364.75	1996.44
34	12	18	6631.76	19.99	5	2169.01	1391.08
35	9	18	6493.04	14.79	5	2144.69	1922.08

表 2：对 LPN 公式应用 CryptoMiniSat 的不同配置。每个 n 值对应 20 个实例。仅显示不可满足实例的验证时间。标记为“T.O.”的列表示超过 3600 秒时限的案例数。

平均值 $n/4$。这些选择与 Crawford 建议的参数相匹配。随后将这些公式分别交由 CryptoMiniSat 和 Tbsat 求解，并将所得证明分别交由 lrat-check 和 frat-rs 进行 Verify。

对于 CryptoMiniSat，我们在 Intel 2xE5-2690v3 CPU 上运行这些测试，每次运行分配 24GB RAM，时间限制为 3600 秒。对于 tbsat，我们在 Ryzen 9 5950x 上运行这些测试，每次运行分配 4 GB，时间限制为 3600 秒，证明大小限制为 2^{30} 子句。证明大小限制是根据我们使用 lrat-check 检查器的经验设定的。允许更大的证明只会生成难以处理的文件，并产生过大而无法检查的证明。

CryptoMiniSat 和frat-rs 性能表2 展示了将CryptoMiniSat 应用于这些公式的几种不同方式的性能，以及相应的 PAR-2 分数[4]。如表所示，对于每个 n 值，不可满足公式的数量介于 8 个（40%）至 16 个（80%）之间。不使用高斯-约旦消元时，CryptoMiniSat 在$n = 27$ 时开始达到时间限制，而当 n 达到 34 时，90% 的公式都会如此。因此，PAR-2 时间趋近于上限值 7200。另一方面，使用高斯-约旦消元且不考虑证明生成时，所有基准测试都能在 20 秒内完成求解。

CryptoMiniSat 与 tbuddy 组合的测量结果，是在启用证明生成的情况下运行求解器得出的，无论公式是否可满足。与纯 CDCL 相比，这种组合能够为规模大得多的问题生成证明。求解器仅在 $n \geq 34$ 上超过了时间限制。与不进行证明生成相比，这会带来显著的性能损失。增加的耗时

[4]PAR-2 分数用于 SAT 竞赛中衡量性能。每个基准测试的分数为解决它所用的秒数；如果超时或内存耗尽，则分数为超时时间（以秒计）的两倍。随后计算所有基准测试的平均分，得出 PAR-2。

N	不可满足	tbsat 高斯-若尔当		lrat-check 平均值（秒）
		P.O.	PAR-2	
20	15	0	0.18	0.38
21	10	0	0.55	0.22
22	13	0	0.37	0.14
23	14	0	0.43	0.11
24	16	0	0.36	0.13
25	11	0	4.49	0.17
26	14	0	1.54	0.21
27	13	0	16.19	0.25
28	8	0	42.51	0.25
29	8	0	69.72	0.44
30	14	2	756.12	0.60
31	15	1	387.51	0.69
32	12	3	1104.10	0.64
33	10	2	792.41	1.15
34	12	4	1489.05	3.24
35	9	4	1565.19	2.32

表 3：在Gauss-Jordan模式下将Tbsat 应用于 LPN公式。对于 n 的每个取值，均有 20 个实例。验证时间仅显示不可满足实例的验证时间。标记为“P.O.”的列表示生成的证明超过 2^{30}子句的实例数量。

可能部分是由于我们的实现集成较为松散。每次使用新的奇偶矩阵行生成原因子句时，Tbuddy 都必须从头开始对奇偶约束的一个子集求和，即使该子集中的许多元素此前已经求和过。不过，要实现更紧密的集成，就必须大幅改写已经复杂的奇偶推理代码。

最后一列显示了在公式不可满足且 CryptoMiniSat能够完成执行的情况下运行 frat-rs的平均时间。总体而言，性能是可以接受的，但frat-rs 检查证明所需的时间可能比生成该证明所需的时间多约 $3\times$ 。在这些证明中，Tbuddy 为其每个证明步骤提供提示，而 CryptoMiniSat 为其中超过 50% 的步骤提供提示，因此检查器性能似乎应该能够得到提升。

Tbsat 和 lrat-check 性能 表 3 展示了 TBSAT 结合 Gauss-Jordan 消元以及 LRAT 证明生成应用于 LPN 公式时的性能。与之前一样，对于每个 n 值，不可满足公式的数量从 8（40%）到 16（80%）不等。一个独立程序会检测 CNF 表示中的奇偶约束，并为 调度表 生成TBSAT，标识每个约束及其对应的子句集合 [5]。 TBSAT 没有任何变量排序启发式，因此用户必须提供变量排序，才能在这些公式上实现良好性能。我们对这些公式采用的排序如下：对至多一个约束进行编码的变量、损坏变量、奇偶约束的辅助变量，最后是解变量。

总体而言，即使考虑到执行平台的性能差异，Tbsat 在这些公式上的运行速度仍显著快于 CryptoMiniSat。另一方面，它的稳定性较差。即使对于小至 $n = 30$ 的公式，它也可能遇到一次合取运算，导致执行过多二叉决策图操作，使证明生成器超出 the

2^{30} 子句的上限。相比之下，CryptoMiniSat 的性能一直非常稳定，直到 n 达到 34 后才开始超出时间限制。然而，对于两个求解器来说，生成的证明都可能达到 100 GB。检查 tbsat 生成的 lrat 证明的速度远快于检查 CryptoMiniSat 生成的 FRAT 证明。tbsat 能够在不导致证明子句计数器溢出的情况下解决的所有问题，运行时间都低于 1200 秒。实际上，如果缩短超时限制，PAR-2 得分还会提高。

4 结论

CDCL 求解器在处理奇偶约束方面存在一个主要弱点。将高斯-约旦消元与 CDCL 相结合已经相当成熟，但到目前为止，这种能力还无法生成不可满足性证明。我们的工作表明，标准的子句证明框架已经足够，而二叉决策图则为奇偶推理与证明生成之间提供了桥梁。我们希望其他研究者能够探索 CDCL(T) 框架，同时保留生成不可满足性证明这一重要能力。

参考文献

[1] S. Baek、M. Carneiro 和 M. J. H. Heule。用于 SAT 求解器与精化器通信的灵活证明格式。载于《系统构建与分析的工具和算法》（TACAS），第 I 部分，LNCS 第 12651 卷，第 59–75 页，2021 年。[2] L. A. Barnett 和 A. Biere。非子句冗余性质。载于《自动演绎会议》（CADE），LNAI 第 12699 卷，第 252–272 页，2021 年。[3] A. Biere、K. Fazekas、M. Fleury 和 M. Heisinger。CaDiCaL、Kissat、Paracooba、Plingeling 和 Treengeling 参加 SAT 竞赛 2020。载于《SAT 竞赛 2020 论文集——求解器与基准测试描述》，计算机科学系报告系列 B 第 B-2020-1 卷，第 51–53 页。赫尔辛基大学，2020 年。[4] R. E. Bryant。TBUDDY：一个可生成证明的 BDD 软件包。载于《计算机辅助设计形式化方法》，2022 年。[5] R. E. Bryant、A. Biere 和 M. J. H. Heule。伪布尔推理的子句证明。载于《系统构建与分析的工具和算法》（TACAS），LNCS，2022 年。[6] R. E. Bryant 和 M. J. H. Heule。使用基于 BDD 的 SAT 求解器生成扩展归结证明。载于《系统构建与分析的工具和算法》（TACAS），第 I 部分，LNCS 第 12651 卷，第 76–93 页，2021 年。[7] R. E. Bryant 和 M. J. H. Heule。使用基于 BDD 的 SAT 求解器生成扩展归结证明。CoRR，abs/2105.00885，2023 年。[8] J. M. Crawford、M. J. Kearns 和 R. E. Schapire。最小分歧奇偶问题作为一个困难的可满足性问题。转载于 https://www.cs.cornell.edu/selman/docs/crawford-parity.pdf，1994 年。[9] L. de Moura 和 N. Bjørner。Z3：一种高效的 SMT 求解器。载于《系统构建与分析的工具和算法》，LNCS 第 4963 卷，第 337–340 页，2008 年。[10] R. Dechter。桶消元：一个统一的推理框架。《人工智能》，113(1–2)：41–85，1999 年。[11] S. Gocht、C. McCreesh 和 J. Nordström。VeriPB：让组合搜索算法值得信赖的简便方法。载于《从约束编程到可信赖的人工智能》，2020 年。[12] S. Gocht 和 J. Nordström。使用伪布尔证明高效验证奇偶推理。载于《AAAI 人工智能会议》，第 3768–3777 页，2021 年。

[13] C.-S. Han和J.-H. R. Jiang。布尔可满足性以单纯形方式遇到高斯消元。载于《计算机辅助验证》（CAV），LNCS第7358 卷，第 410–426 页，2012年。[14] M. J. H. Heule、W. A. Hunt、M.Kaufmann和N.D. Wetzler。命题证明的高效、经验证检查。载于《交互式定理证明》，LNCS第10499 284 卷，2017 269 年，第 – 页。[15] T.Jussila、C.Sinz和A. Biere。带量化的符号 SAT 求解的扩展归结证明。载于《可满足性测试理论与应用》（SAT），LNCS第 4121 卷，第 54–60 页，2006 年。[16] O. Kullmann。关于扩展归结的一般化。离散应用数学，96- 97：149–176，1999 年。[17] T. Laitinen、T.Junttila 和I.Niemelä。使用完整奇偶推理扩展子句学习 SAT 求解器。载于《人工智能工具国际会议》，第 65–72 页。IEEE，2012 年。[18] C.-M. Li。DLL过程中等价文字传播。离散应用数学，130(2)：251–276，2003年。[19] J. Lind-Nielsen。BuDDy：二叉决策图软件包。丹麦技术大学信息技术系，1996年。[20] G. Pan和M.Y. Vardi。可满足性求解中的搜索与符号技术。载于《可满足性测试理论与应用》（SAT），LNCS第3542 卷，第 235–250 页，2005 年。[21] T.Philipp 和A.Rebola-Pardo。用于异或推理的 DRAT 证明。载于《人工智能中的逻辑》，LNAI 第10021 卷，第 415–429 页，2016 年。[22] C. Sinz 和 A.Biere。用于合取 BDD 的扩展归结证明。载于《俄罗斯计算机科学研讨会》（CSR），LNCS 第3967 卷，第600–611 页，2006 年。[23] M.Soos、S.Gocht 和K.S.Meel。有色、分离且惰性的 CNF-XOR 求解及其在计数和采样中的应用。载于《计算机辅助验证》（CAV），LNCS 第12224 卷，第463–484 页，2020 年。[24] M.Soos、K.Nohl 和 C.Castelluccia。将 SAT 求解器扩展至密码学问题。载于《可满足性测试理论与应用》（SAT），LNCS第 5584 257 卷，2009244 年，第 – 页。[25] Y. K. Tan、M.J. H. Heule 和 M.O.Myreen。cake_lpr：CakeML 中经验证的传播冗余检查。载于《系统构造与分析的工具和算法》（TACAS），第二部分，LNCS 第12652 卷，第 223–241页，2021 年。[26] G. S. Tseitin。关于命题演算中推导的复杂性。载于《推理自动化：计算逻辑经典论文 2：1967–1970》，第466–483 页。Springer，1983 年。[27] A. Urquhart。归结的困难实例。J.ACM，34(1)：209–219，1987 年。[28] N. D. Wetzler、M. J. H.Heule 和W. A. HuntJr。DRAT-trim：使用富有表现力的子句证明进行高效检查和裁剪。载于《可满足性测试理论与应用》（SAT），LNCS 第 8561 卷，第 422–429 页，2014 年。