利用奥卡姆剃刀原理进行统计模式学习

Justin Talbot*
justin. talbot@databricks.
com Databricks
美国加利福尼亚州旧金山市

Daniel Ting*
dting.tr@gmail.
com 美国华盛顿州
西雅图市 Tableau
研究机构

摘要

经过合理规范化的数据库模式能够提高数据的可解释性、减少数据规模并增强数据完整性。然而，现实世界中的数据集通常以非规范化状态存储或共享。我们研究了如何自动为非规范化表创建良好模式的问题，将其视为一个无监督机器学习问题，必须从数据中学习出最优模式。这与过去基于规则的方法不同，后者侧重于将数据规范化为标准形式。我们基于奥卡姆剃刀原理定义了一个原则性的模式优化标准，该标准对噪声具有鲁棒性且可扩展——允许用户轻松指定所期望的模式属性。我们为此标准开发了一种高效的学习算法，并通过实证表明，该算法比以往的工作快 3 到 100 倍，并且生成的模式质量更高，错误更少。

CCS 概念

· 计算方法学！机器学习； · 信息系统！数据库设计与模型。

关键词

数据库规范化、信息论、贝叶斯学习

ACM 参考格式：

贾斯汀·塔尔博特和丹尼尔·廷。2022 年。利用奥卡姆剃刀原理进行统计模式学习。载于 2022 年国际数据管理会议（SIGMOD’22）会议录，2022 年 6 月 12 日至 17 日，美国宾夕法尼亚州费城。美国纽约：ACM，14 页。https://doi.org/10.1145/3514221.3526174

1 简介

模式规范化的好处是众所周知的：它减少了数据冗余，降低了存储需求；它使数据完整性约束更容易实施；而且当模式中的表与有意义的领域概念相对应时，它还能提高数据的可解释性和可用性。然而，通常一个单一的非规范化表更容易共享和查询。因此，在开始分析或数据建模项目时，一个常见的挑战是将非规范化数据集分解为一个良好的规范化模式。手动完成这一工作可能会很困难且耗时。

为解决这一问题，我们致力于从非规范化输入表中自动恢复无损、规范化的雪花型模式。我们提出了一种概率机器学习模型以及相应的无监督学习算法，与现有方法相比，该算法在鲁棒性、质量和速度方面具有优势。具体来说，

它从一个广泛的搜索空间中选择出定量上最优的模式，自然地挑选出更有可能包含可解释实体和关系的模式。

它对数据中的随机性具有很强的鲁棒性，能够处理假阴性（因数据中的噪声而被掩盖的功能依赖关系）和假阳性（数据中存在但与现实世界中的依赖关系不对应的函数依赖关系）。

（3）它提供了一种有原则的方法，用于优先选择具有理想属性的模式，例如存在简单的候选键。

它可能比现有的方法快几个数量级。

我们的方法通过直接优化模式质量的原则性度量来实现这一目标。这与现有的模式规范化方法不同，后者侧重于满足由规范范式强加的技术条件，但不一定能生成简洁、组织良好的模式。此外，满足这些条件需要在数据中发现函数依赖关系，这是一项成本高昂的任务，而我们的方法则避免了这一过程。

在本文中，我们做出了若干方法论和理论方面的贡献。我们展示了如何将奥卡姆剃刀原理（即最简单的解释往往是最好的解释）应用于模式学习问题，从而得出一个有原则的模式质量度量标准。然后，我们将这种方法嵌入到一个贝叶斯概率框架中，使其易于针对其他理想的模式属性（例如简洁的主键）进行优化。结合这些贡献，我们开发了一种高效的算法来寻找最优模式。我们在真实世界的数据集上进行了实验，结果表明在鲁棒性、质量和速度方面都有显著的改进。

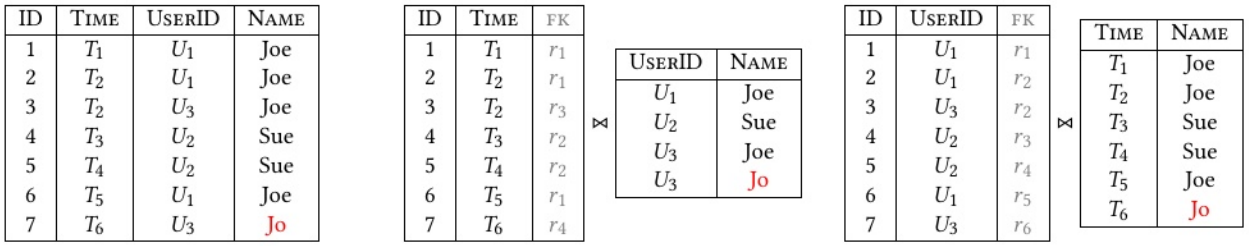
2 相关工作与挑战

手动模式规范化[4, 6, 10]的标准方法依赖于两阶段过程：（1）指定关系中成立的函数依赖（FD），（2）利用这些函数依赖将关系分解为所需的范式。Papenbrock 和 Naumann [30] 利用函数依赖发现算法的最新进展来实现这一过程的自动化，生成无损的雪花模式模式，且处于 Boyce-Codd 范式（BCNF）。Kenig 等人[20]放宽了雪花模式的要求，通过查找近似的多值依赖，然后利用它们生成有损的无环模式。

我们发现这种两阶段方法存在两个实际问题。首先，数据集中少量的错误可能会掩盖功能依赖关系，从而无法恢复重要的实体；相反，偶然的依赖关系也常常会随机出现，导致规范化后的模式中包含的表与现实世界中的实体并不对应（图 1）。后一个问题并不令人意外，因为

*This work was done while the authors were employed at Tableau Software.

SIGMOD’22, June 12–17, 2022, Philadelphia, PA, USA
 ACM ISBN 978-1-4503-9249-5/22/06...$15.00 https://doi.org/10.1145/3514221.3526174



ID	Time	UserID	Name
1	T_1	U_1	Joe
2	T_2	U_1	Joe
3	T_2	U_3	Joe
4	T_3	U_2	Sue
5	T_4	U_2	Sue
6	T_5	U_1	Joe
7	T_6	U_3	Jo

(a) Denormalized table, $\mathcal{D}$ (error on line 7)

ID	Time	FK
1	T_1	r_1
2	T_2	r_1
3	T_2	r_3
4	T_3	r_2
5	T_4	r_2
6	T_5	r_1
7	T_6	r_4

⋈

UserID	Name
U_1	Joe
U_2	Sue
U_3	Joe
U_3	Jo

(b) Desired normalization, despite error

ID	UserID	FK
1	U_1	r_1
2	U_1	r_2
3	U_3	r_2
4	U_2	r_3
5	U_2	r_4
6	U_1	r_5
7	U_3	r_6

⋈

Time	Name
T_1	Joe
T_2	Joe
T_3	Sue
T_4	Sue
T_5	Joe
T_6	Jo

(c) Semantically meaningless normalization

图 1：从数据依赖关系中学习模式时遇到的困难示例。给定 D，我们希望学习到（b）中的模式（外键列引用第二张表中的行号），但第 7 行中的错误（“Jo”而非“Joe”）掩盖了 UffffffID 和 Nffffff 之间的函数依赖关系。此外，由于 Tffffff 的基数较高，它无意中函数确定了 Nffffff（但不是 UffffffID），这可能导致两阶段规范化方法生成（c）中的模式，错误地将 UffffffID 和 Nffffff 分离。

潜在函数依赖的数量以及因此产生的意外依赖的数量会随着列数的增加呈指数增长。尽管先前的研究开发了用于查找近似依赖关系的方法[16, 21, 36]以及忽略虚假依赖关系的方法[7, 23, 24]，但如何校准这些方法及其超参数以使所得函数依赖关系对自动模式规范化有意义，这一问题仍未得到解决。当前的模式规范化方法[30]试图通过应用启发式方法来优先考虑函数依赖关系以缓解意外依赖关系的问题；然而，这导致生成的模式质量低于我们方法所找到的模式质量。其次，尽管算法不断进步[1, 12, 16, 22, 26, 29, 34, 35]，但枚举数据派生依赖关系在计算上仍然成本高昂。在两阶段方法中，搜索阶段发现的绝大多数函数依赖关系在后续的规范化阶段都不会被使用，因此大部分计算都是浪费的。

我们转而采用信息论方法来确定最优模式。先前的研究[2, 20, 21, 24, 30, 36]使用信息论方法来衡量和利用列之间的依赖关系。然而，这些研究都基于传统假设，即数据中的所有行都是从单一固定的基础分布中独立抽取的。而在我们的研究中，规范化模式中的每个表都对应于不同的随机分布。非规范化表中的行是从这些共享分布中抽取的。由于改变其中一个基础分布会改变多行，所以这些行是相关的。我们的模型可以看作是利用列和行之间的依赖关系来确定更好的模式分解。从统计学角度来看，我们的方法类似于 Chow-Liu 算法[9]，学习具有树形结构的图形模型。但它与 Chow-Liu 以及过去关于结构学习的文献[11, 18]不同，这些文献都假定行是独立且同分布的。

我们的工作有一个附带效果，即生成输入数据的压缩表示，因此也与表压缩方法相关。先前的工作对列的联合分布进行建模[14, 31]，并捕获细粒度的依赖关系[15, 17]以生成更高效的编码，而我们的方法则捕获对模式规范化有用的依赖关系。

3 模式学习概览

与基于字段驱动的模式学习方法不同，我们将模式学习表述为一个无监督机器学习问题，直接搜索最优模式。这需要三个组成部分：（1）包含合理模式的优化搜索空间；（2）评估模式质量的原则性目标函数；（3）能够高效解决优化问题的算法。

第 4 节描述了我们的优化搜索空间——概念雪花模式，在这种模式中定义了表，但未标识主键。该空间被证明等同于列的多级聚类。为了评估模式质量，第 5 节展示了如何运用奥卡姆剃刀原理，基于最小描述长度（MDL）推导出一个原则性的目标函数，该函数自然地奖励那些能更简洁地捕捉数据中实体的模式。然后，我们将其发展为一个贝叶斯模型，为用户提供了一种公式化的方法，将其他理想的模式属性纳入目标函数。

然而，由此产生的优化问题并非易事。优化是在一个离散空间中进行的，其规模随列数呈超指数增长，且目标函数的计算成本高昂。第 6 节提供了一种高效的算法，该算法结合了分支定界法、动态规划和贪婪启发式方法。这使得中等规模的问题能够得到精确优化，而大规模问题则能获得近似解。第 7 节表明，我们不仅获得了比现有技术更高质量的结果，而且运行时间快了多达两个数量级。为避免在阐述中出现过多的技术细节，我们将所有证明都放在了附录中。

4 优化搜索空间

给定一个非规范化表，我们的目标是寻找一个规范化模式，其中其表代表可解释的真实世界实体。搜索空间必须足够广泛以涵盖此类模式，但也要受到一定限制以使优化具有可行性。无环模式在数据建模方面具有许多良好特性[5]。然而，此类模式的空间很大，并且在实际中，当存在噪声或缺失行时，它们很少允许无损连接分解。相反，我们专注于更受限制的雪花模式空间，该空间在先前的工作中也有使用[30]。所谓雪花模式，是指其实体关系图形成一个具有多对一关系的 n 元有根树，从父表到子表。与一般的无环模式相比，雪花模式的受限空间即使对于有噪声的数据也总是允许进行简单的无损连接分解。此外，雪花模式

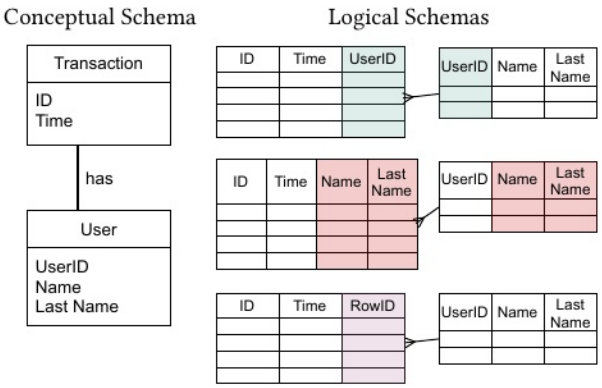


图 2：概念模式和逻辑模式。单个概念雪花模式（左）可根据主键的选择（着色部分）表示为多个逻辑模式（右上）。为避免选择主键的成本，我们的方法仅在使用物理行索引作为键（右下）的模式上进行搜索。此行主键列是隐式的，无需进行实体化。

模式通常可以使用连接表来对无环模式中的多对多和一对多关系进行建模，从而缩小表示能力上的差距。最后，我们注意到对多对多关系进行反规范化会导致数据量急剧增加；因此，我们认为来自无环模式的反规范化表在实际中出现的可能性要小于来自雪花模式的反规范化表。

在基于函数依赖的模式规范化中，结果模式的主键是在规范化过程中确定的。用于拆分表的函数依赖的左侧定义了该表的主键。相比之下，在搜索过程中我们不会从候选键集中确定主键。为了生成无损连接分解而不确定主键，我们只需使用行号来链接表（图 2）。这种方法基于两个考虑因素。首先，对于一个实体，简洁的主键可能在未规范化表中不存在，或者可能因噪声而损坏。使用行号作为主键的替代品使我们的算法能够稳健地处理这些情况。其次，不识别主键简化了搜索空间。由于我们无需复制输入列来创建外键 - 主键对，输入表中的每个列在输出模式中仅出现一次。这意味着雪花模式可以被视为输入列的层次分区。在第 6 节中，我们利用这一特性来创建一种高效的算法以找到最优模式。（请注意，之后可以运行一个主键检测算法[19]，以向我们的模式添加主键。）

虽然我们已对搜索空间加以限制，以使优化问题更易于处理，但该空间仍然很大。由于我们的雪花模式空间等同于输入列的层次分区空间，因此搜索空间在列数上呈超指数增长。

设 Γ 为优化函数搜索空间中的雪花模式集合。模式空间呈超指数增长，且 log|Γ| = Ω（< log<）。

这比可能的功能依赖关系的空间要大得多。因此，与之前的工作相比，我们名义上增加了问题的难度。然而，这种优化框架允许更高效的算法和更好的规范化。

5 优化目标

接下来，我们将从两个角度阐述我们的优化目标——信息论角度和统计学角度。信息论角度简单明了，与人们熟悉的数据库物理设计概念紧密相关。受奥卡姆剃刀原理的启发，我们认为能够以更简单方式描述数据的模式通常更优。为了量化这种简单性，我们采用存储数据所需的比特数。由于模式会影响数据的物理布局，
因此，对于每个模式，都可以根据其数据在无损分解时的大小来衡量其复杂程度，从而选择能使该度量最小化的最优模式。

我们不采用特定的编码方案或压缩算法来衡量数据大小，而是使用信息论中的熵，因为它是数据大小的一个下限。这被纳入到一个最小描述长度的度量中，该度量包括分解后的数据大小、压缩代码的大小以及任何必要的元数据。

这种信息论方法将模式学习问题转化为一个优化问题，但它本质上是启发式的，并且只关注模式设计的一个好处——减少存储。因此，我们还提出了第二种视角，使用概率生成模型来形式化和推广该方法，使我们能够扩展目标以涵盖良好模式设计的其他好处，例如存在简单的主键。

特别是，我们的信息论方法是统计建模和模型选择问题的一个特定实例。由于熵的定义为 E?(-) log2 ?（-），信息论方法需要对潜在的概率分布做出假设并进行估计。通过首先确定每个模式的潜在生成模型，我们可以将其扩展为贝叶斯模型，在该模型中，每个模式都被分配一个概率，表示给定模式为正确模式的置信度。

5.1 信息论解释

在我们的搜索空间中，每一种可能的雪花模式 S 都对应于输入数据 D 的一种无损连接分解。我们的目标是选择最能捕捉数据内部结构（即实体和关系）的模式。应用奥卡姆剃刀原理的直觉表明，对应于更简单、更紧凑的连接分解的模式能更好地捕捉输入数据的结构。我们可以将这种直觉形式化为一种最小描述长度（MDL）目标，其形式为：

$$J_{MDL}(\mathcal{S}|\mathcal{D}) := Length(\mathcal{D}|\mathcal{S}) + Length(\mathcal{S}) \quad (1)$$

其中，！4=6C⌘（D|S） 表示数据在由 S 确定的连接分解中存储的长度，而！4=6C⌘(S) 表示模式的长度。此目标倾向于简洁表示数据的连接分解，同时对过于复杂的模式进行惩罚。

为了使这一目标具体化，！4=6C⌘（D|S） 表示模式的连接分解中所有表的总大小。我们假定数据列和外键列都以理想熵进行存储。

编码；因此，每列的大小 2 即为其经验熵（2）乘以对应表格中的行数 =C：

s the number of rows in its corresponding table, n_t:

$$Length(\mathcal{D}|\mathcal{S}) := \sum_{t \in Tbls(\mathcal{S})} \sum_{c \in Cols(t)} n_t \cdot H(c) \quad (2)$$

模式的长度，4=6C⌘(S)；是存储连接分解中每个列的压缩码的总成本（在 -

（包括任何外键列）

cluding any foreign key columns):

$$Length(\mathcal{S}) := \sum_{t \in Tbls(\mathcal{S})} \sum_{c \in Cols(t)} \beta \cdot |\Omega_c| = \beta \left(\sum_{t \in \mathcal{S}} n_t \right) + const \quad (3)$$

其中，Ω_2 是列 2 中的唯一值集合，V 表示存储一个代码条目的成本。简化为模式中的总行数，这是基于这样一个事实，即只有外键的域在模式之间可能不同。

虽然此度量标准基于输入数据的压缩长度，但我们的目标不仅仅是压缩。基于 MDL 的关键见解在于，能够很好地压缩数据的模式必然也能简洁地捕捉到数据中的重要结构。

5.2 概率解释概述

在上一节的信息论解释中，我们的方法找到了使数据量最小化的模式。在本节中，我们给出一个等效的概率解释，在这种解释下，我们的方法找到了生成输入的非规范化表的正则化似然性最大的模式。

这种框架有助于更好地理解 MDL 目标所作的假设，为将模型选择问题转化为可高效解决的纯优化问题提供了一种途径，并为纳入关于真实模式的先验信息提供了一个框架。我们注意到，基于优化的模型选择加上复杂度惩罚并非新事物，有时可被视为对完全贝叶斯方法的近似[33]。

我们的目标是根据给定的数据生成可能模式的良好的后验分布 P（S|D）。与信息论中的 MDL 目标类似，可以通过最大化此后验分布来找到最优的、最大后验概率（MAP）模式。构建此后验分布需要两个组成部分，一个是生成模型，它定义了给定模式下数据的似然性 P（D|S），另一个是先验分布 c(S)，它倾向于具有良好属性的模式。根据贝叶斯法则，所得的后验分布为 P(S|D) = P(D|S)c(S) 。

要创建生成模型，我们可以将雪花模式视为一个分层图形模型，该模型编码了各列集合之间的条件依赖关系（图 3）。这导致了一个直接的生成过程，其中父表是在其子表的条件下生成的。然而，要形成一个完整的分层生成模型，还需要一组额外的参数 d 来指定如何生成每个表中的数据。由于在模式学习中只有模式本身是感兴趣的，这些 d 被称为无用参数。

因此，在我们的概率解释中，我们必须（1）指定完整的生成分布 P（D|S,d），（2）处理这些无用参数 d，以及（3）指定先验分布 c(S)。在第 5.3 节中，我们表明方程 1 中给出的 MDL 目标对于上述每个问题都有一个特定的解决方案。然后在第 5.4 节中，我们利用我们的概率框架将目标扩展以包含有用的先验信息。

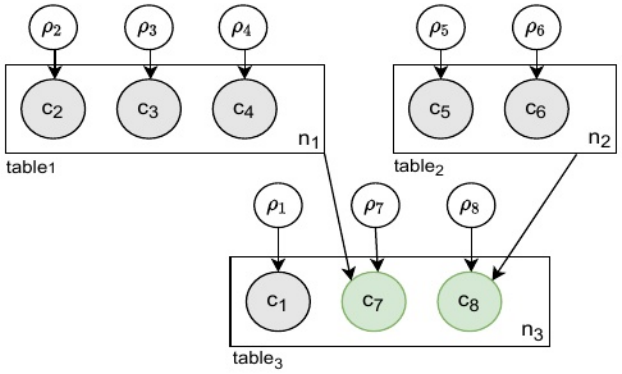


图 3：具有 3 个表、6 个数据列（c1 - c6）和 2 个外键列（c7 和 c8）的模式的图形模型。干扰参数 d 定义了每个列中值的分布。通过首先生成表 1 和表 2，然后独立地为列 - .，1 绘制 n3 个条目，可以从该模型生成一个非规范化表

来自分布 d 的 1 个样本

从 2-1 和 table2 中获取 1 和 n3 元组其概率分别为 d7 和 d8。这里，表 C 中的第 29 列与归一化数据集中的 - C9 相同。

5.3 概率解释详情

要生成一个非规范化表， - ，从模式 S 出发，我们将其视为该模式作为分层图形模型，如图 3 所示。（我们在此使用 -⊠ 来区分从我们的模型中抽取的随机非规范化表与观测数据 D）。我们的生成过程是为模式 S 中的每个表 C 抽取一个表实例 - C；然后将这些表连接起来以创建 - 。

这个数字

对于每一行 =C，在模型中 C 被视为固定值，每一列 2 的域 Ω 也是固定的 2 。未知参数是每一列 2 的概率分布 d 2，这些列可以是数据列也可以是外键列。我们通过以下数据生成过程绘制一张表 - C：

- 对于 C 中的每一列数据 2，分别独立地绘制 =C 指定的 - 从分布 d 中得出的条件期望值。换句话说，C

  88

  3

  -82 ⊠d_2 为 8 ⊠=C 。
- 对于 C 中的每个外键列 2，同样绘制 -82C 883 对应 8 ⊠= C 。

左箭头 d2

反规范化表，-，由所有 -C 连接而成。

此表的似然函数可以写成：

$$p(\tilde{X}|\mathcal{S}, \rho) = \prod_{t \in Tbls(\mathcal{S})} \prod_{c \in Cols(t)} \prod_{i=1}^{n_t} \rho_c(X_{ic}^t). \quad (4)$$

虽然使用外键的表示方法使得 - C 中的所有列都是独立的，但图 3 中的图形模型表示了生成非规范化表 - 的过程。

如果表 1、表 2 和

如果分别用 3 代表客户、店铺和交易，那么只有在知晓交易行中客户的和店铺的信息后，才能填写该交易行的信息。

最大化此似然函数的对数等同于最小化 MDL 目标的一部分。

在上述生成模型下，模式 S 的最大对数似然值为

$$-J_{MLE}(\mathcal{S}|\mathcal{D}) := \sup_{\rho} \ell(\mathcal{S}, \rho; \mathcal{D}) = -Length(\mathcal{D}|\mathcal{S}) \quad (5)$$

其中 d = {?$_2$ （·|S）} $_{2\,2}$⊠>；B(S) 对每一列的经验分布进行编码。

在此，对数似然函数由√（S, d； D）：= log?（D|S, d）定义，并将条件概率（它是数据的函数）转换为参数的函数。对干扰参数d进行最大化处理，使其成为仅关于模式S的目标函数。对于每一列2，使该对数似然函数最大化的d$_2$是将数据规范化为模式S后该列的观测经验分布。例如，在图1b的模式中，Nffffff上的分布将“Jo”的概率赋值为1/4，而不是图1a中未规范化表中给出的1/7。

这种生成式表述使我们能够将模式学习视为贝叶斯估计问题，并允许我们注入关于正确或最佳模式的先验信念。如果模式 S 的先验分布不依赖于干扰参数，即 c0(S, d) / c(S)，那么我们就能得到一种更通用形式的 MDL 目标函数。

$$J(\mathcal{S}|\mathcal{D}) := -\max_{\rho} \log_2 p(\mathcal{S}, \rho|\mathcal{D}) = J_{MLE}(\mathcal{S}|\mathcal{D}) - \log_2 \pi(\mathcal{S}) \quad (6)$$

当先验概率仅取决于模式的长度时，上一节中的 MDL 目标函数 “$_{⊠!}$” 就可以恢复为：log⊠ c(S) = !4=6C⌘(S) + 2>=BC 。

5.4 构建有用的先验条件

先验概率使我们能够调整目标函数，以偏爱具有除描述长度短之外的有用属性的模式。然而，方程 6 中的形式过于笼统，既未提供如何编码有用属性的指导，也未说明如何调整先验概率，还允许存在会使优化变得困难的先验概率。在本节中，我们提供了生成先验概率的模板，这些模板能够实现目标函数的高效优化，并具有有助于从业者设置参数的有意义的解释。

给定模式 S 和表 C ∈ B(S)，设 ⊠⊠$_C$ 表示通过连接 C 及其后代所重建的表。为了实现高效

为了优化目标，我们考虑形式为对数先验的函数：

$$\log_2 \pi(\mathcal{S}) = \sum_{t \in Tbls(\mathcal{S})} g(parent(t), \tilde{\mathcal{D}}_t). \quad (7)$$

我们的学习算法对列执行分层分区，这种形式的先验仅利用分区步骤前后立即可用的信息。我们注意到，这种形式的先验包括形如 log2 c(S) = $_{C\ 2)1;\ B(S)\ 6(C)}$ 的先验，它仅考虑单个表，而不考虑模式中的关系。

我们的基本设置将负对数先验 log2 c 类似于优化问题中的软约束进行处理。一个惩罚目标函数，用于抑制表格 C 具有属性 P，可表示为 $_{"!}$⊠ (S|D) + _I（C 2 P），其中 0 和 I（C 2 P）−是指标当 C 满足 % 时取值为 1，否则取值为 0 的函数。由于每当 C 具有不期望的属性 % 时，目标值就会增加，因此具有此类模式的选项处于劣势，不太可能被选中。

我们考虑了三种此类形式的有用对数先验，它们分别用于不同的目的。这三种先验（1）对具有不良属性的表进行惩罚，（2）抑制分裂，（3）鼓励理想的分裂：

$$\log_2 \pi(\mathcal{S}) = -\sum_{t \in Tbls(\mathcal{S})} \gamma \cdot \mathbb{I}(t \in \mathcal{P}) \cdot n_t \quad \text{(Table penalty)}$$

$$\log_2 \pi(\mathcal{S}) = -\sum_{t \in Tbls(\mathcal{S})} \gamma \cdot \mathbb{I}(\tilde{D}_t \in \mathcal{P}) \cdot n_{pa(t)} \quad \text{(Discourage)}$$

$$\log_2 \pi(\mathcal{S}) = \sum_{t \in Tbls(\mathcal{S})} \tau \cdot \mathbb{I}(t \in \mathcal{P}) \cdot n_t \log_2 n_t \quad \text{(Encourage)}$$

其中 W > 0 且 g ∈ [0, 1) 是用户指定的权重。请注意，对于分裂抑制先验，指标只能检查关于 ⊠⊠$_C$ 的属性，其中不包含关于进一步分裂出哪些列的信息，而不是关于 C 的属性。

行数项将参数 W 和 g 进行缩放，以便在 MDL 和贝叶斯框架下具有直观的解释。带有 W 的惩罚项相当于在表或其父表中添加一个熵为 W 的列。鼓励缩放 g 消除了创建新表的开销中的分数 g。通过减少开销来奖励某种属性的行为使其有别于惩罚方法，并且无法用软约束来表达。尽管可以通过在某些属性未满足时施加惩罚来模拟鼓励行为，但这种惩罚会增强 MDL 先验，这可能并非所期望的。这是因为默认的按行数进行惩罚的行为正是方程 3 中 MDL 惩罚所执行的。

我们注意到，其他先前的形式也能支持高效的优化。例如，在某些情况下，当某个属性得到满足时施加一个常数惩罚，或者使用随表大小次线性增长的其他缩放方式可能是合理的。同样，指示函数也可以用一个平滑函数来替代。

给定一组先验值，log2 c$_8$，其形式如方程表中的惩罚项所示，我们可以将它们组合起来形成我们的最终先验值。

$$\log_2 \pi(\mathcal{S}) = \sum_{\mathcal{P}} \log_2 \pi_i(\mathcal{S}) + const. \quad (8)$$

我们将这种形式的先验分布定义为规范先验，并附加一个约束条件，即鼓励先验的缩放参数之和满足 g$_{C>C}$ = $_8$ g$_8$ ∈ [0, 1)。稍后，在定理 4 中，我们将证明此类先验分布可导致一种高效的优化算法。以下示例展示了上述先验分布能够捕捉到的特性：

示例 1：具有共同前缀的列名很可能来自同一个源实体。如果这些列不同时出现，可以将其编码为表惩罚。如果具有共同前缀的列为 F，则可以通过设置 I$^{(C\ 2\ P)}$ = I（|F \ ⊠>；B(C)| 62 {0, |F |}） 来检查该属性。

示例 2：经常一起查询的列很可能来自同一个实体。此场景与示例 1 类似，但为 F 使用了不同的列集。在这种情况下，将缩放参数 W 设为与这些列一起被查询的次数成比例可能是合理的。这使得工作负载或性能信息能够影响模式的选择。

示例 3：表很可能具有简单的候选键。这可以通过鼓励先验和抑制先验的组合进行编码。抑制先验会惩罚创建没有简单候选键的表。鼓励先验则会奖励

即使与父表近乎一对一的关系，创建一个具有简单候选键的表，也不会显著减少数据冗余。

6 学习算法

Algorithm 1: SSLearn

Input: Denormalized table, $\mathcal{D}$
Output: Optimal snowflake schema, $\mathcal{S}$

Memoized Function `Optimize`($\mathcal{D}$):
  // Initialize queue and upper bound with empty schema
  let q = new PriorityQueue(orderby = $J(\mathcal{S})$)
  q.push({$\mathcal{S}$: $\emptyset$, C_{parent}: [], C_{child}: [], C_{tail}: $Cols(\mathcal{D})$})
  let upper = `UpperBound`($\emptyset$, $\mathcal{D}$)
  while q is not empty **do**
    let {$\mathcal{S}$, C_{parent}, C_{child}, C_{tail}} = q.pop()
    // If all columns have been partitioned, return
    if C_{tail} = [] **then**
      return $\mathcal{S}$
    // Apply bounds to prune search space
    if `LowerBound`($\mathcal{S}$, $\Pi^{+}_{C_{tail}}\mathcal{D}$) > upper **then**
      continue
    // Otherwise, branch on next column in table
    [c_{head} | C'_{tail}] = C_{tail}
    // Branch with c_{head} partitioned to parent table
    C'_{parent} = append(C_{parent}, c_{head})
    $\mathcal{S}_0$ = `Normalize`($\mathcal{D}$, C'_{parent}, C_{child})
    q.push({$\mathcal{S}_0$, C'_{parent}, C_{child}, C'_{tail}})
    // Branch with c_{head} partitioned to child table
    C'_{child} = append(C_{child}, c_{head})
    $\mathcal{S}_1$ = `Normalize`($\mathcal{D}$, C_{parent}, C'_{child})
    q.push({$\mathcal{S}_1$, C_{parent}, C'_{child}, C'_{tail}})
    // Use prefix solutions to improve upper bound
    upper = min(upper, `UpperBound`($\mathcal{S}_0$, $\Pi^{+}_{C'_{tail}}\mathcal{D}$)),
     `UpperBound`($\mathcal{S}_1$, $\Pi^{+}_{C'_{tail}}\mathcal{D}$))

Function `Normalize`($\mathcal{D}$, C_{parent}, C_{child}):
  let $\mathcal{D}_{parent}$ = $\Pi^{+}_{C_{parent}}(\mathcal{D})$, $\mathcal{D}_{child}$ = $\Pi_{C_{child}}(\mathcal{D})$
  if $\mathcal{D}_{parent}$ = $\mathcal{D}$ or $\mathcal{D}_{child}$ = $\mathcal{D}$ **then**
    return $\mathcal{D}$
  else
    $\mathcal{S}_{parent}$ = `Optimize`($\mathcal{D}_{parent}$)
    $\mathcal{S}_{child}$ = `Optimize`($\mathcal{D}_{child}$)
    return `LinkWithFK`($\mathcal{S}_{parent}$, $\mathcal{S}_{child}$)

优化我们的目标需要在超级指数级大的搜索空间中进行组合搜索。为了使这一问题变得可行，首先，我们提出了一种全局学习算法 SSLfffffff，它将搜索结构化为列的分层划分，这使我们能够结合动态规划和分支定界方法来进行高效搜索，通常无需枚举所有可能的模式。其次，我们将 SSLfffffffff 扩展为一种任何时算法，能够在任何时间点返回良好的（即便不是最优的）解决方案。

6.1 SSL

SSL 算法通过递归地对输入表 D 的列进行二分划分来枚举雪花模式的模式空间（见图 4）。每次划分都会在提议的雪花模式中创建一个新的子表。子表会进行去重处理，而父表则不会，因此父表和子表之间的多对一连接可以无损地恢复输入表。对父表和子表进行递归划分可以生成任意的雪花模式。我们可以通过在每一步找到最优的列划分来找到最优的雪花模式。正式地：

Tfffffffffff 3. 设 （S|D） 是方程 6 中形式的目标函数，具有方程 7 中形式的先验条件。将 ☒>；B(D) 分为两个不相交的集合，记为 $C_{?0A4=C}$ 和 $C_{2⌘8;3}$，并记保持重复项的投影为 D ? 0A4=C = Π+C?0A4=C D，去重复项的投影为 D $_{2⌘8;\ 3}$ = $\Pi C_{2⌘8;}$ $_{3}$ *Projection* $\mathcal{D}_{child}$ = $\Pi_{C_{child}}\mathcal{D}$. Then,

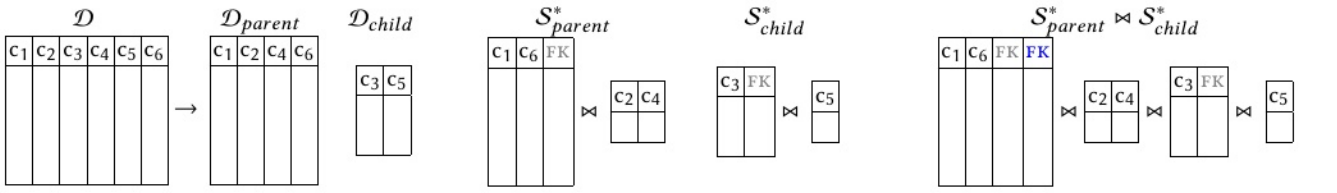
$$\min_{\mathcal{S}} J(\mathcal{S}|\mathcal{D}) = \min_{C_{parent},C_{child}} \Big(w(\mathcal{D}_{parent},\mathcal{D}_{child}) + \min_{\mathcal{S}_{parent}} J\left(\mathcal{S}_{parent} \mid \mathcal{D}_{parent}\right) + \min_{\mathcal{S}_{child}} J\left(\mathcal{S}_{child} \mid \mathcal{D}_{child}\right) \Big) \tag{9}$$

其中 F 是一个不依赖于 S 的函数。

算法 1 中简要概述了这种最小化的实现。Optimize 使用分支定界搜索来找到 ☒>；B(D) 的最优划分 $C_{?0A4=C}$ 和 $C_{2⌘8;3}$（方程 3 中的外部最小化）。给定一个划分，Normalize 将 D 分割成两个子表 $D_{?0A4=C}$ 和 $D_{2⌘8;3}$。如果划分退化，则递归终止，并返回仅包含 D 的单个表模式。否则，它递归地在每个子表上调用 Optimize 以找到各自的最优雪花模式（方程 3 中的内部最小化），然后创建一个外键将它们链接起来，返回 D 的雪花模式分解。

在每次调用 Optimize 时，必须探索的列分区数量众多，这使得暴力枚举变得不可行。相反，我们使用分支定界法来限制搜索空间。我们的分支定界搜索通过为 ☒>；B(D) 的越来越长的前缀找到最优解来实现。这些前缀解用于找到最优完整解的上界和下界，我们利用这些界来修剪搜索空间。

在“优化”中，搜索从将两个分区 $☒_{?0A4=C}$ 和 $☒_{2⌘8;3}$ 初始化为空列表开始，并将包含搜索尚未探索的列的 $☒_{C08;}$ 初始化为 ☒>；B(D)。在搜索的每个节点，我们从 $☒_{C08;}$ 中取出第一个列 $2_{⌘403}$ 并分支，在搜索优先级队列中添加两个新节点，将 $2_{⌘403}$ 分别添加到 $☒_{?0A4=C}$ 或 $☒_{2⌘8;3}$ 中。当 ☒ $_{C08;}$ 为空时，所有列都已分区，所得分区即为最优解。（由于优先级队列是按照我们的目标排序的，任何进一步的解都必须是非最优的。）在每个节点，我们还为前缀 $\Pi_{☒^{+}{}_{?0A4=C}[☒\ 2⌘8;}$ $_{3}$ D 找到最优解。这与未探索列的宽松界限 Π☒+C08； D 相结合，以生成完整解的上限和下限：



(a) Partition the columns of $\mathcal{D}$ into two subtables—$\mathcal{D}_{parent}$ and $\mathcal{D}_{child}$. Deduplicate the latter.

(b) Recursively find the optimal snowflake schema, $\mathcal{S}^*$, for each subtable (introducing foreign keys as needed).

(c) Create a new FK (blue) linking the root table of $\mathcal{S}^*_{parent}$ to the root of $\mathcal{S}^*_{child}$, creating a snowflake for $\mathcal{D}$.

图 4：SSLfffffffff 通过对输入表进行递归二分分区来构建雪花模式。递归过程在任何分区都不能改善目标值时终止。在递归的每一层，都使用分支定界法来找到使 最小的分区，从而为 D 生成最优的雪花模式。

下界：任何最优前缀解都为完整解提供了一个简单的下界。我们可以通过以下定理稍微收紧这个下界：

假设目标函数形式如方程 6 所示，且具有第 5.4 节中描述的典型先验 c。设 S☒、 S_C☒be 分别为目标函数 （·|D）、（·|D_C）的最小值点。那么，对于任何 C ☒☒>；B(D

$$J(\mathcal{S}^*|\mathcal{D}) \geq J(\mathcal{S}^*_C|\mathcal{D}_C) + (1-\tau_{tot}) \sum_{c \in C^C} |\Omega_c| \log_2 |\Omega_c|. \quad (10)$$

上限：包含所有列的任何模式都为最优解提供了一个简单的上限。我们可以通过将未探索的列添加到根表中，将任何前缀解扩展为包含所有列的解，从而得到以下上限。设 S☒、SC☒ 分别为目标函数 （·|D）、（·|D_C） 的最小值。分别。然后，对于任意的 C ☒☒>；B(D)：

$$J(\mathcal{S}^*|\mathcal{D}) \leq J(\mathcal{S}^*_C|\mathcal{D}_C) + \sum_{c \in C^C} n \cdot H(c) \quad (11)$$

其中（2）是输入数据 D 中第 2 列的熵，而 = 是根表中的行数。由于列域和熵只需计算一次并存储起来，因此这些界限的评估成本很低。

随着优化过程的推进——为越来越长的前缀找到最优解——这些上界和下界会变得越来越紧。在搜索的早期阶段找到紧的界可以显著提高算法的运行时间。为此，我们将列按照从高到低的熵值进行排序。通过将通常包含结构重要列的高熵列放在前面，将对优化目标贡献较小的低熵列放在后面，搜索能够更快地揭示最优模式的结构，并收紧上界和下界。

请注意，在我们的递归优化中，相同的子问题可能会多次出现。根据定理 3，这些子问题的解可以组合起来得到全局解，因此我们采用自顶向下的动态规划，将 Optimize 的结果进行记忆化处理。这极大地提高了算法的效率。

最后，我们的方法通过对同一表进行重复的二分划分来生成一般的 n 叉树。这会引入一些冗余，因为多个划分序列可能会产生相同的最终多路划分。由于这种排序不影响目标，我们使用分配给每个分区的最小列索引来施加一个规范排序。

6.2 任意时间算法的扩展

尽管上一节所述的分支定界加动态规划算法相对高效，但随着列数的增加，运行时间仍呈指数增长，如图 10 所示。为解决这一问题，我们将展示如何对其进行扩展，使其能够在任何时刻返回一个良好的近似解。这还有第二个好处，即可以收紧上界，从而允许搜索进行更多的剪枝。请记住，之前的两阶段方法必须先找到函数依赖关系（这是算法中成本最高的部分），然后才能进行规范化，因此无法提供早期的解决方案。

方程 11 中的上限值对应于一个近似解，其目标值计算成本较低，但由于只是简单地将 ☒ C08 中的所有列添加到根表中，所以该解可能较差。为了找到更好的近似解，我们偶尔会运行一个成本更高的贪心分配过程，该过程按顺序将 ☒ C08; 中的列分配给当前模式中能使目标值最小化的表。由此产生的模式的目标值可用作改进后的上限值。

由于这种贪婪分配过程运行成本较高，因此只有在模式结构可能发生改变时才会触发。我们跟踪与方程 11 中上限相关的部分模式，并且仅在对 Optimize 的根调用中此上限发生变化时才运行贪婪分配。如果算法在达到搜索的时间限制之前未完成，则返回迄今为止找到的最佳解决方案，该方案由前缀列的最优解加上剩余列的贪婪放置组成。

请注意，这种贪心算法将剩余的列分配给部分最优模式中的现有表；它不会创建新表或更改模式中的关系。因此，需要穷举搜索组件来找到全局最优模式。

7 实验

为了评估我们所提出的统计模式学习（SSL）方法的有效性，我们在 Rust 语言中实现了一个单线程版本的算法。我们将我们的结果与 Papenbrock 和 Naumann（P&N）[30] 的现有最先进方法进行了比较，该方法是并行化的，并用 Java 语言实现 [28]。

7.1 实验装置

我们所有的 SSLfffffffff 实验均在一台配备双 CPU 英特尔至强 E5-2630 v3 @ 2.40GHz 的机器上运行，该机器拥有 16 个物理核心和

我们使用了 192GB 内存的 CentOS Linux 7 系统。对于 P&N，我们在更快的双 CPU Intel Xeon Silver 4114 @ 2.20GHz（20 个物理核心和 192GB 内存）上运行了一些耗时较长的实验。由于 SSLfffffffff 是一种任何时刻算法，我们报告找到最优解的时间（此时提前终止即可得到最优解）以及找到该解并验证其最优性的总时间。P&N 只在完成时返回模式，因此我们仅报告其总时间。

如第 5.4 节所述，我们的算法可以通过信息先验利用额外的信息，例如查询历史记录。为了与 P&N 进行公平比较，我们未利用这一优势；在以下实验中，两种算法使用相同的输入数据集。我们确实包含了一个优先项，倾向于具有简单候选键的规范化表；P&N 的基于启发式的方法也做出了类似的假设，实际上生成了一个启发式构建的“黄金集”函数依赖。具体而言，我们使用 MDL 目标（公式 1），设置 V = 1，并为具有简单候选键的表添加先验。依照第 5.4 节中的示例 3，我们添加了一个惩罚先验，W = 10，以惩罚没有简单候选键的表，以及一个鼓励先验，g = 1/2，以鼓励将具有简单键的表拆分出来。这些选择具有自然的解释。W 的选择通过增加存储外键的成本来惩罚没有简单候选键的表，每个条目需要多 10 位。g 的选择将具有简单键的主键存储成本减半。我们发现，调整 MDL 惩罚项的权重 V 对所学模式影响甚微，因为来自数据的证据随数据规模线性增长，而先验知识则不然。我们发现，从定性角度来看，同时使用非零的 W 和 g 对所学模式的质量有显著影响。然而，当将 W 的参数从 10 变化到 15 时进行的有限实验并未产生定性上的不同结果。我们没有探索改变 g 的情况。

最后，为了避免在简单情况下采用贪婪算法（第 6.2 节）所带来的开销，因为在这种情况下它并没有带来多少好处，我们仅在计算出前 15 列的最优模式之后才执行贪婪分配。

7.2 数据

我们在 TPC-H 以及多个真实世界的数据集上评估了我们的方法，这些数据集包括 Musicbrainz [13] 和来自 CTU 关系学习库 [25] 的 5 个数据集。这些数据集已经过规范化处理，可作为良好模式的基准。

对于每个标准化的数据集，我们构建包含数据集中部分或全部表的非规范化表。在表 1 中，我们总结了这些非规范化表，指出了每个表的源数据集、包含的表数量、总列数以及用于非规范化原始模式的连接图的形状：星型和雪花型连接图仅包含具有单个根节点的一对多关系；无环图包含多对多连接，而有环图包含连接图中的环。Musicbrainz 非规范化表与先前的最新技术[30]中使用的相同；对于此数据集，我们知道源表，但不知道连接图的真实情况。

之后，我们还报告了在北卡罗来纳州选民登记数据集 [27] 上的结果，该数据集以单个表格形式分发，没有真实模式。此数据集包含 820 万行和 71 列。

7.3 学习到的模式的质量

为了评估所学模式的质量，我们将它们与真实情况下的规范化模式进行比较。由于我们不知道有适用于此任务的现有模式相似性度量标准，因此我们使用编辑距离来衡量一个人可能需要执行的编辑操作数量，以使所学模式与真实情况下的模式相匹配。为了定义编辑距离，我们考虑模式上的三种编辑操作：（1）合并两个表，（2）将一个表拆分为两个表，（3）在两个表之间移动一列。编辑距离是使所学模式与原始模式匹配所需的最少编辑操作数。如果原始模式中的实体与所学模式中的实体具有相同的数据列（而非外键列），则称所学模式与原始模式匹配。更确切地说，如果从所学模式中的表到原始模式中的表存在单射映射，并且对于所学模式中的每个表 C 及其在原始模式中的对应表 C0，C 的列是 C0 列的子集，则称模式匹配。虽然已有不少关于自动计算各种树编辑距离的研究工作 [8, 32]，但我们未能找到能计算我们所定义的编辑距离或具有类似语义的编辑距离的现有算法。因此，我们在实验中手动计算了编辑距离。这在我们的实验中是可行的，因为距离较小，且大多涉及易于解决的合并操作。学习到的模式和代码可在[3]获取。

由于我们的主要任务是在数据中识别有意义的实体，而我们的许多数据集都是从非 Snowflake 架构生成的，所以我们不比较学习到的架构和原始架构的图形结构；我们只比较实体本身。

由于 P&N 生成的逻辑模式包含主键，因此我们通过删除外键信息将 P&N 模式转换为概念模式，如图 2 所示。在某些情况下，P&N 可能会复制一列以在多个表中形成主键；例如，在图 6 中，c_id 出现在多个表中，这些表对应于真实的 Track 和 Recording 表。在这种情况下，我们将该列分配给能最大程度展现 P&N 优势的表，即与底层模式编辑距离最小的表。

我们还按每种编辑操作的数量来分解编辑距离。由于从一个模式转换到另一个模式存在多种编辑序列，我们采用优先合并、其次列移动、最后列拆分的编辑序列。这种偏好倾向于选择需要较少选择的操作。选择两个表进行合并被认为比选择要拆分出哪些列的子集更容易，前者最多有二次方数量的选择，而后者则有指数级数量的选择。

在反规范化表中，对于属于原始模式中不同表（例如外键 - 主键对）的重复列，分配错误将被忽略；仅基于数据的算法无法区分这些列。原始模式中通过一对一关系连接的表也被视为单个表，因为仅根据其值的分布无法区分它们。

7.4 质量结果

图 5 展示了 SSLfffffffff（蓝色）和 P&N（红色）在所有数据集上每种类型的编辑数量以及总的编辑距离。SSLfffffffff 学习到的模式与原始模式更为接近，其总的编辑距离仅为 P&N 的 $1/5^{C⌘}$ 。P&N 极易受到

	输入			已学习模式中的表数量以及与原始模式的编辑距离 SSLflflflfl P&N									
数据集	汤匙	列	模式	汤匙	合并	分裂	移动	距离	汤匙	合并	分裂	移动	距离
TPC-H	8	52	雪花	10	2	0	0	2	15	7	0	1	8
Musicbrainz	11	113	未知的	11	0	0	2	2	20	10	1	8	19
PKDD 金融	6	45	非环的	5	0	1	0	1	9	4	2	0	6
MovieLens（演员）	5	13	非环的	3	0	0	2	2	5	0	0	1	1
MovieLens（用户）	5	14	非环的	4	0	0	1	1	5	0	0	1	1
FNHK	3	22	循环的	5	1	0	0	1	10	5	0	0	5
StackOverflow（帖子）	4	44	雪花	4	0	0	0	0	25	18	0	0	18
StackOverflow（投票数）	5	48	非环的	8	1	0	0	1	10	5	0	0	5
CCS 事务	4	18	星星	5	1	0	1	2	8	4	0	0	4

表 1：学习到的模式的质量。通过编辑距离衡量，SSLflflflfl 生成的模式更接近真实情况。

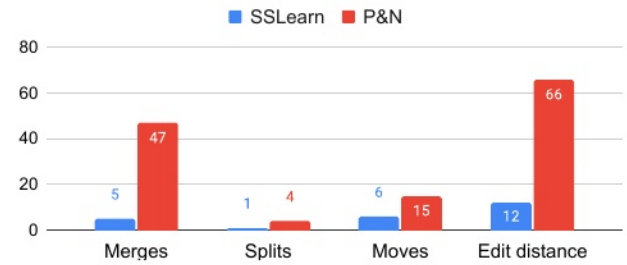


图 5：表中各数据集相对于原始规范化模式的单个编辑次数及总编辑距离 1（数值越低越好）

过度拆分导致生成的表数量远超原始模式中的表数量。例如，TPC-H 的原始模式有 8 个表，SSLflflflfl 学习到的模式有 10 个表；合并其中 2 对表即可还原原始模式。相比之下，P&N 学习到的模式有 15 个表；需要合并 7 对表，并在表之间移动 1 列才能还原原始模式。表 1 中对此进行了详细说明，表明即使底层模式不满足模型假设的分层雪花结构，SSLflflflfl 的优势依然存在。在 9 个案例中，P&N 仅在 1 个案例中返回了质量稍高的结果。

此外，我们注意到，在一些被标记为错误的情况下，SSLflflflfl 实际上揭示了具有实际意义的真实世界实体。例如，CCS 数据集包含加油站的交易记录。SSLflflflfl 生成了一个语义合理的表格，识别出了加油站连锁品牌。然而，这在原始的加油站模式中并未建模，所以我们将其归类为错误。同样，在 TPC-H 中，SSLflflflfl 从零件表中分离出了零件制造商——这是一种语义合理的规范化，但在原始模式中并不存在。并且它使用了一个额外的表来对 TPC-H 定义中的跨列依赖关系进行建模，即所有状态为“O”的订单行的发货日期都在 1995 年 6 月 18 日之前，所有状态为“F”的订单行的发货日期都在 1995 年 6 月 18 日或之后。相比之下，图 6 和图 8 表明，P&N 的过度拆分行为通常不会产生具有实际意义的实体。

我们注意到，对于非雪花型模式，例如在 Stack-Overflow（投票）数据集中，由于这两种算法都存在雪花型模式的约束，SSLflflflfl 和 P&N 都会学习到原始模式中不存在的连接表。然而，即便在这种情况下，SSLflflflfl 仍能学习到源模式中的实体。

为了定性地展示所学模式质量的差异，图 6 的左侧展示了 Papenbrock 和 Naumann [30] 提供的 MusicBrainz 数据集的结果，右侧则是 SSL 的结果。树中的每个节点代表所学模式中的一个表。为简化显示，我们使用 l_* 表示一组以 l_ 为前缀的列名。

在这两种情况下，方法都学习到了一个雪花模式，其中包含一个根连接表，以捕获原始模式中存在的艺术家、地点、发行标签和曲目之间的多对多关系。然而，在这种情况下，P&N 极度过度划分了模式，生成了 20 个表，而原始模式只有 11 个表（额外的表用红色三角形标记；这些需要合并到父表中以重建原始模式）。它还未能正确放置大量低基数列（用黄色突出显示；这些需要移动到正确的位置以重建原始模式）。相比之下，SSLflflflfl 生成了一个近乎完美的雪花模式；只有两个非常低基数的列位置错误。此外，尽管这些列 a_type 和 a_edits_pending 属于艺术家表，但它们在功能上依赖于较小的 artist_credit 表的主键 ac_id。我们还注意到，这两种方法将 artist_credit 放在了两个不同的表下。在这种情况下，这两种放置方式都可以被认为是正确的，因为 artist 和 artist_credit 以及 release_group 和 artist_credit 之间都存在外键 - 主键关系，然而，SSLflflflfl 将其与较小的 artist 表关联起来，这可以说是一种更具语义意义的关系。

7.5 噪声的影响

为了测试算法对数据集中噪声的鲁棒性，我们选取了 TPC-H 数据集，并随机将其中 0.02% 的条目替换为同一列中随机选取的值。这相当于破坏了 1% 的行。这种更具挑战性的情况需要花费更多的时间，因此我们仅使用了非规范化 TPC-H 数据集中 20% 的订单。

对于此有噪声的 TPC-H 版本所学习到的模式如图 7 所示。尽管存在噪声，SSLflflflfl 仍然能够

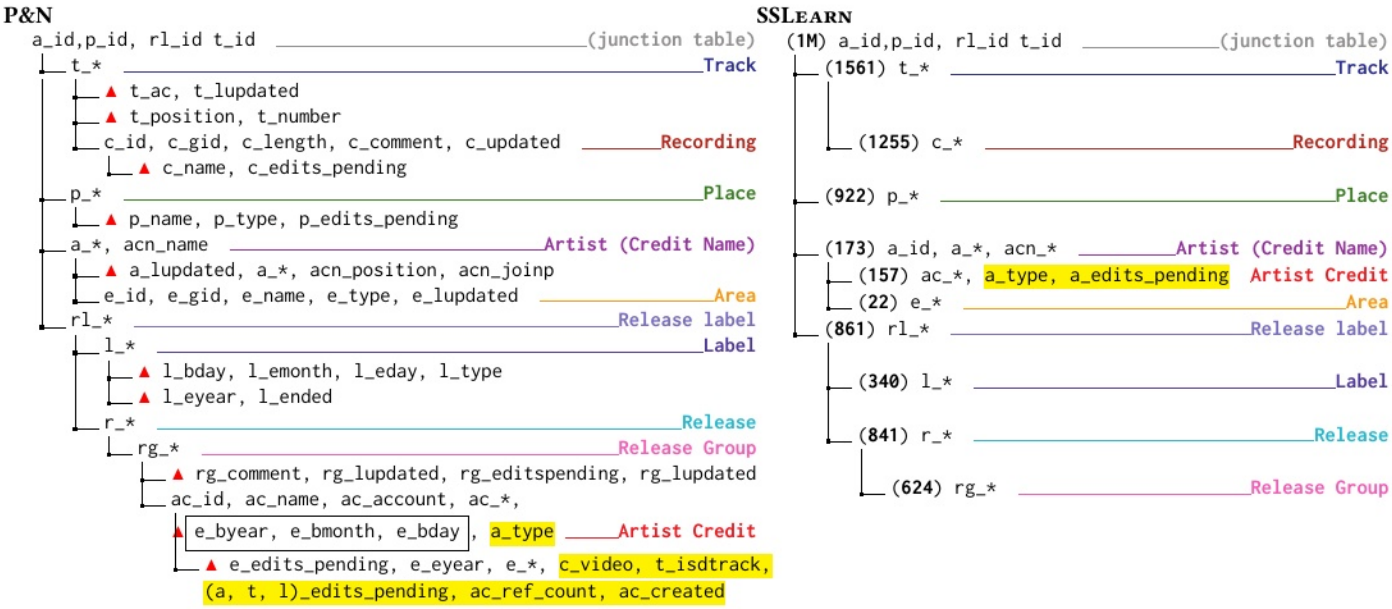


图 6：在 MusicBrainz 上学习到的模式的比较。前缀 * 通配符替换多个此类形式的列，（a,b）_c 表示 a_c、b_c。我们标注了将学习到的模式转换为真实模式所需的编辑：框中的实体应进一步拆分，带有 N 的实体应与父实体合并，突出显示的列应移动到模式中的不同分支。P&N 由于数据中语义不重要的函数依赖关系而严重过度划分模式，并且错误地放置了低基数列。相比之下，SSLflflflfl 非常准确地学习了原始模式，仅错误地放置了两个非常低基数的列。

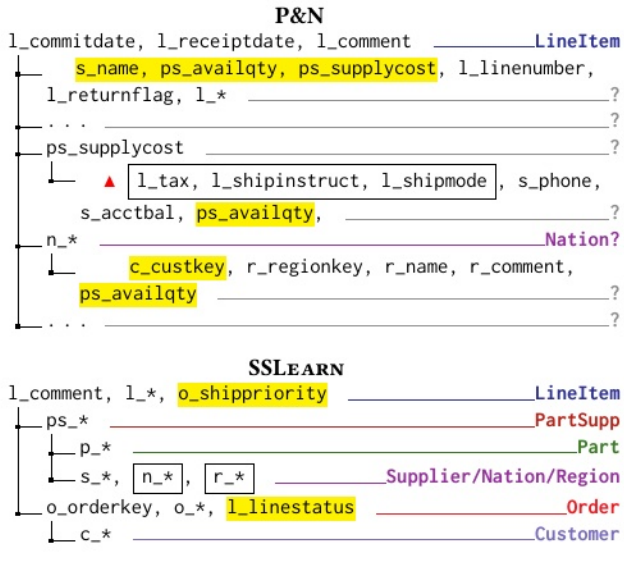


图 7：从含噪声的 TPC-H 中学习到的模式。上部：P&N 生成的 23 个表中的 7 个表。很少有表能映射到语义上有意义的实体。下部：SSLflflflfl 几乎完全恢复了 TPC-H 的真实模式；只有两列位置错误，且 Suppliers（供应商）、Nations（国家）和 Regions（地区）被放在了一个表中。

恢复了 TPC-H 中的大部分结构。它将供应商、国家和区域表合并为一张表，并将 o_shippriority 分配给订单项表，将 l_linestatus 分配给订单表。不过，核心结构已成功识别。

相比之下，我们发现 P&N 学习到的模式中几乎没有有意义的结构。在这种情况下，P&N 生成了 23 个表。然而，与无噪声的情况不同，在无噪声的情况下，额外的表可以简单地合并以重建原始模式，而在这种有噪声的情况下，许多表是由来自原始模式中多个表的属性组成的。图 7 展示了 P&N 学习到的部分模式；对这个模式进行任何直接的转换都无法恢复原始的真实模式。因此，在存在少量噪声的情况下，P&N 无法揭示出语义上有意义的结构。

7.6 真实世界的非规范化数据集

我们还在北卡罗来纳州选民登记数据集上展示了 SSLflflflfl 的效果，该数据集以单个表格形式分发，且没有真实的模式。图 8 展示了 P&N 和 SSLflflflfl 的前几层模式。P&N 学习到了许多高基数的表，并由于偶然的功能依赖关系将选民人口统计列分散在模式中。相比之下，SSLflflflfl 将列组织成简洁且语义可解释的模式，其中包含一个包含选民特定属性的根表、一个越来越具体的地理区域层次结构以及一组低基数的字典表。

7.7 性能

除了能找到更高质量的模式外，图 9 还表明，SSL 的速度比 P&N 快得多，快了多达两个数量级。

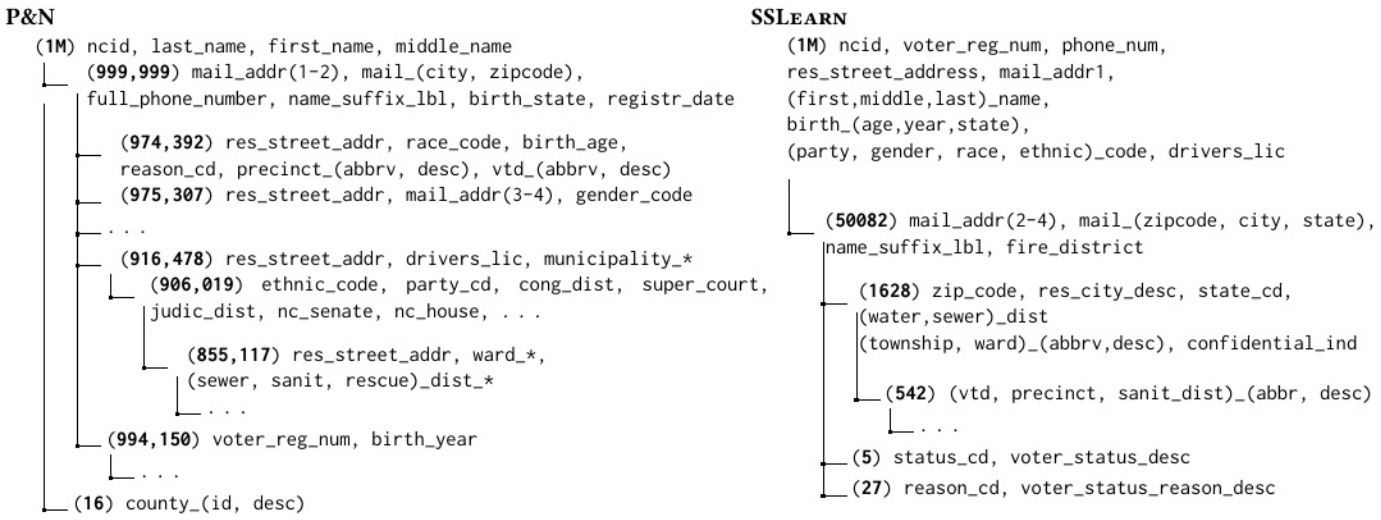


图 8：在北卡罗来纳州选民登记数据集上学习到的模式。（a,b）_（c,d）表示 4 个属性：a_c、b_c、a_d、b_d。P&N 对表进行了过度划分，并将人口统计信息分散在整个模式中。SSLflflflfl 生成了一个简洁且具有语义意义的模式，其中选民人口统计信息位于根表中，并且存在地理层次结构。

数据集	输入 行	列	模式	SSLflflflfl 解决方案时间（分钟） 总时间	P&N 总用时	加速 解决方案	总计
CCS 交易	一千	18	星星	0.00	0.00	0.01	30.00	30.00
MovieLens（演员）	148 千字节	13	非环的	0.01	0.01	0.03	6.57	6.57
MovieLens（用户）	1.1 米	14	非环的	0.03	0.03	0.18	6.89	6.89
FNHK	2.3 米	22	循环的	0.80	0.90	1.88	2.35	2.09
StackOverflow（帖子）	213 开尔文	44	雪花	3.80	4.00	12.37	3.25	3.09
PKDD 金融	130 万	45	非环的	1.20	1.70	12.32	10.26	7.25
StackOverflow（投票数）	3.1 M	48	非环的	1.00	1.70	226.35	226.35	133.15
TPC-H（抽样 + 噪声）	1.1 米	52	雪花	2.40	132.90	364.78	151.99	2.74
北卡罗来纳州选民	100 万	71	未知的	33.18	33.54	547.80	16.51	16.33
Musicbrainz	100 万	113	未知的	2.50	1125.88	1775.73	710.29	1.58
TPC-H	6 米	52	雪花	20.00	24.00	1973.77	98.69	82.24

表 2：在我们尝试的所有情况下，SSLflflflfl 都更快，速度提升最高可达两个数量级（总加速比列）。

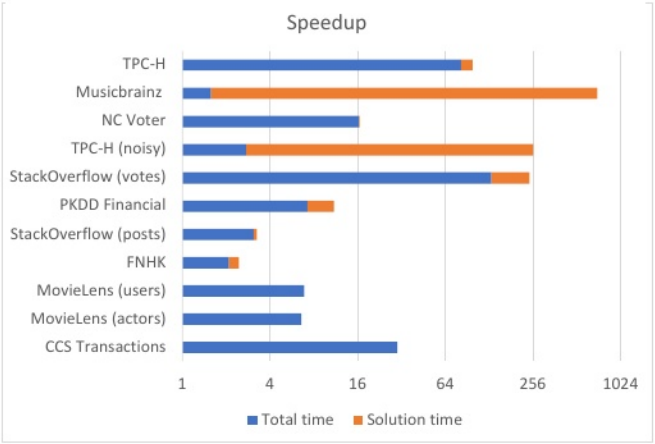


图 9：按数据集划分的加速比（P&N 运行时间 / SSLearn 运行时间）。蓝色柱形表示 SSLflflflfl 所花费的总时间，橙色柱形表示 SSLflflflfl 找到解决方案时所花费的时间。

在大型数据集上的规模。我们的单线程算法在 25 分钟内对 TPC-H 的全部 52 列进行了详尽搜索，而 P&N 即便在 20 个物理核心上运行也需超过 1.5 天。SSLflflflfl 在我们测试的所有数据集上都更快。详细性能结果见表 2。

此外，尽管搜索可能需要很长时间，但找到最优解的速度可能会快得多。在某些情况下，大部分时间都花在验证早期候选解是否为最优解上。对于 MusicBrainz 数据集，最终解在 2.5 分钟时找到，而验证该解为最优解则花费了 19 个小时。这一特性还使我们能够构建在限定时间内执行的方法。由于 SSL 在所有数据集上均在 35 分钟内找到解，将运行时间限制在 1 小时内，即可得到一个在限定时间内执行并返回最优结果的程序。然而，求解器在验证当前解是否为最优解时所花费的长时间段，使得很难预测找到良好模式的合适截止时间。我们发现，找到最优解的时间在很大程度上取决于列的搜索顺序。如果，

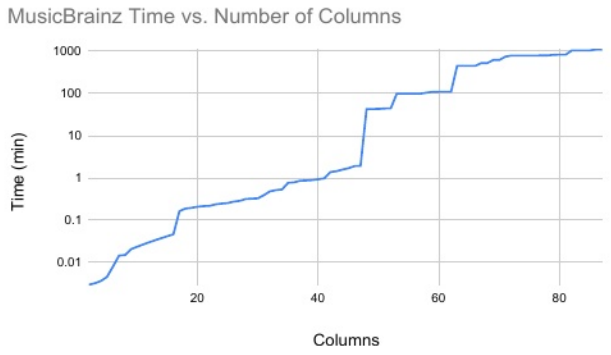


图 10：在前 G 列上学习最佳模式所需的时间。

对于每一张表，候选键都较早出现在搜索过程中，因此解决方案很快就找到了。

尽管 SSLflflflfl 比 P&N 快得多，但图 10 显示，在 MusicBrainz 数据集上，我们的算法在进行穷举搜索时，其运行时间仍随处理的列数呈指数增长。因此，始终能够保持良好的候选解是该算法的一个有用特性。

8 分析

第 5 节中开发的统计框架使我们能够分析我们的方法是如何运作的。特别是，它使我们能够研究我们的方法何时选择拆分表格，以及列独立性假设在我们的生成模型中所起的作用。这有助于我们理解我们的方法相对于相关的信息理论工作的独特属性和优势。

考虑模式 S 中的表 C。对于列集 C ☒ ☒>；B(C)，设 S0 是通过将 C 中的列 C 分离出来形成去重表 C 0 而得到的模式。

考虑模式 S、S0，其中唯一的区别在于对于某些表 C ∈ B(S) 及其列 C ☒ B(C)，S0 将 C 分裂，并将列 C 放入新表 C0 中。考虑来自 C 中列 C 的元组，并将元组的经验联合分布记为 ?☒$_{C^0}$。

模式学习倾向于选择分裂模式 S0 当且仅当 *we choose S′ if and only if*

$$n_t \, KL\left(\hat{p}_{t'} \,\middle\|\, \prod_{c \in C} \hat{p}_c\right) > n_{t'} \sum_{c \in Cols(t')} H(c) = Size(t') \qquad (12)$$

其中 ?☒$_{C^0}$，？ ☒$_2$ 分别为 C0 中各行的经验分布 - .

或者第 2 列。！ 是用于衡量分布差异的 Kullback-Leibler 散度。

左边表示通过用外键替换原始表中多个列中的冗余信息所节省的空间。右边表示创建新子表的成本。

8.1 与近似FD度量的比较

我们可以将其与用于检测近似函数依赖的信息论度量进行比较。分数信息度量[23]用于检验函数依赖 - ! 是否存在，其定义为

$$F(X;Y) := \frac{H(Y) - H(Y|X)}{H(Y)} = \frac{KL(\hat{p}_{XY} || \hat{p}_X \hat{p}_Y)}{H(Y)} \qquad (13)$$

其中，？ ☒. 和 ?☒. 分别表示 . 中所有列和 - 以及 .. 中所有列的经验分布。如果对于某个常数 2，(-;.) > 2 或者等价地，如果 !(?☒. ||?☒.？ ☒.) > 2 （.），则拒绝 - ! . 是函数依赖的零假设。如果检测到函数依赖，则可以使用由函数依赖驱动的模式规范化方法将 -，. 中的列拆分到另一个表 C 0 中。

比较方程 12 和 13，主要的相似之处在于都使用了 KL 散度来衡量依赖关系。它将 C0 中列的联合分布 (?☒$_{C^0}$ 或 ?☒. 完全依赖的模型与将列或列组视为独立的模型进行了比较。这也证明了我们模型中列独立性的假设是合理的。与近似 FD 度量一样，即使列并非真正独立，独立性假设也能提供有用的依赖性度量。

我们的度量方法也有一些关键的不同之处和优势。首先，公式 12 考虑了表的大小 =$_C$,=$_{C^0}$ 。这一关键的不同之处使得我们的方法能够敏感地检测到在多对一连接中自然出现的重复。相比之下，仅依赖于值的分布的熵度量只能间接地衡量重复的数量。因此，我们的目标不是使用通用的相关性度量，而是专门针对检测连接。其次，我们的方法更易于应用。它不需要选择阈值或确定适当的相关程度。第三，与近似 FD 度量 [23] 不同，我们的方法不需要额外的偏差校正来避免过多的误报。

9 讨论与未来工作

我们的模式学习方法在多个应用中有着明显的应用价值，包括数据分析和探索。通过发现离散值列中具有意义的聚类和层次结构，它能够帮助数据分析师在数据探索过程中更好地理解列之间的关系。将我们的模型扩展以处理连续值列是未来工作的一个潜在方向。未来工作的另一个可能方向是将我们的方法扩展为学习无环模式。第 7.4 节的实验表明，在某些情况下，我们的方法能够学习到模式中具有意义的实体，但会创建存在多对多关系的连接表。作为后期处理步骤移除这些连接表可以恢复原始的无环模式。另一个未来方向是将我们的方法扩展为显式地对数据中的噪声进行建模，例如通过使用噪声模型对相似行进行聚类。这将提高统计模式学习的鲁棒性。

未来算法研究的一个方向是采用马尔可夫链蒙特卡罗（MCMC）方法。这将允许使用更灵活的先验分布，这些先验分布无需具有递归分解，并且能够更好地探索在穷举搜索接近尾声时所搜索的列的分配情况。不过，这会失去动态规划所获得的效率。

统计模式学习并不强制所学模式遵循标准范式。不过，现有的能够保证规范化为标准范式的现有方法可以作为后处理步骤运行。由于我们的方法可以对比原始表规模更小且列数更少的子表进行规范化处理，因此能够加快这些方法的运行速度。

10 结论

我们提出了一种新的模式学习范式，其直接通过优化良好属性来驱动，而非假定遵循某种规范范式自然就能得到良好的模式。其中最为重要的是最小描述长度这一属性，它量化了奥卡姆剃刀原理。因此，即便作为一种没有真实基准可比的无监督学习方法，我们的模型也有一套原则性的方法，能够从非规范化数据集中挑选出有意义的真实世界实体。通过开发贝叶斯统计框架，该方法具有了可扩展性，能够将其他理想属性纳入优化标准。重要的是，我们通过开发一种高效的分支定界算法结合动态规划，使这种范式下的计算变得可行。我们的实验表明，这种统计建模与高效优化算法的结合，能够在更短的时间内产生比现有方法质量更高且更抗噪的结果。

A PROOFS 证明；证据；定理的证明

TflflflflflflflflfI 1 flflflflfl. 我们表明，即使是更简单的星型模式类也已经是超指数级的。考虑将 < 列进行划分的方法数量。这是贝尔数 ☒< ，其渐近满足 log ☒< = = log= + $（= log log=） 。 ☒

Tflflflflflflfl 2 flflflflfl. 我们首先展示经验熵是“D;C8=><80;（=, ? 1, . . . ?: ）”的负最大对数似然值。设 - 是从此分布中抽取的样本。类别概率的最大似然估计值显然就是经验概率 ?☒: = -: /=。 以 2 为底的最大对数似然值为 ((3D)
mum log-likelihood is
每一列都是独立的，并且服从多项分布。由于列是独立生成的，所以最大对数似然函数为 ☒ =: = G =: log☒ ?☒ = = (-)

$$\sup_{\rho} \ell(\mathcal{S}, \rho|\mathcal{D}) = \sum_{t\in Tbls(\mathcal{S})} \sum_{c\in Cols(t)} -n_t H(c) = -\sum_{t\in Tbls(\mathcal{S})} Length(t)$$

□

设 C 为分配给 S 中根节点的某个子节点 C 及其后代的列。设 S_C 表示以 C 为根的子树，设 S0 为将 S_C 删除后的 S。简单的代数运算得出

$$J(\mathcal{S}|\mathcal{D}) = w(\mathcal{D}_{C^C}, \mathcal{D}_C^U) + nH(c, t_C) + g(\mathcal{D}, \mathcal{D}_t)$$
$$+ J\left(\mathcal{S}_0 \mid \mathcal{D}_{C^C}\right) + J\left(\mathcal{S}_t \mid \mathcal{D}_C^U\right)$$

root to table t. Minimizing this objective gives
where2 5 : 是作为外键指向 C 的列。我们可以 takeF(D$_{C☒}$， D*$_C$) = = (25:): + 6（D， D☒ 其中 $S_{25:}$ 是将根表与表 C 相连接的外键列。使该目标函数最小化可得

$$\min_{\mathcal{S}} J(\mathcal{S}|\mathcal{D}) = \min_{C\subset Cols(\mathcal{D})} \min_{S_0, S_1} \Big(w(\mathcal{D}_{C^C}, \mathcal{D}_C^U)$$
$$+ J\left(S_0 \mid \mathcal{D}_{C^C}\right) + J\left(S_1 \mid \mathcal{D}_C^U\right)\Big)$$
$$= \min_{C\subset Cols(\mathcal{D})} \Big(w(\mathcal{D}_{C^C}, \mathcal{D}_C^U)$$
$$+ \min_{S_0} J\left(S_0 \mid \mathcal{D}_{C^C}\right) + \min_{S_1} J\left(S_1 \mid \mathcal{D}_C^U\right)\Big)$$

☒

由于给定 C 时 F 是常数，所以最后一个等式成立。S0 的变化不会影响 S1，反之亦然，因为它们的列是不相交的。

考虑最小化值 S☒。设 T1 表示仅包含 C☒ 中列的表的集合，T2 表示包含 C☒ 中某些列的其余表的集合。目标函数可以写成表成本之和。我们首先考察成本

$$\sum_{c\in Tbls(t)} H(c) \geq (1-\tau_{tot}) n_t \sum_{c\in Tbls(t)} H(c)$$

T1 中的表的数量。对于 C2T1，

$$\geq (1-\tau_{tot}) n_t \sum_{c\in Tbls(t)} |\Omega_i| \log_2 |\Omega_i|.$$

第一个不等式源于这样一个事实：联合分布的熵至多等于各边缘分布熵之和。

对于 T0 中的表 C，记 C0 为在添加任何来自 C ☒ 的列之前的表，=0 为 C0 中的行数。我们考虑只添加了来自 C ☒ 的一列 2 的情况。添加额外的列只会增加额外的行，因此我们将表分为 3 部分：（1）C 的子表，等于 C0，包含部分行和除 2 以外的所有列；（2）剩余的行和除 2 以外的所有列；（3）列 2 中的值。使用一个编码对 C 进行编码的成本下限为分别使用每个部分的最优编码对这 3 个部分进行编码的成本之和。使用最优列编码对部分（1）进行编码的成本恰好是编码 C0 的原始成本。添加 2 后先验概率的变化为 $\Delta_c = g_{C>C}$ （=0 log2 =0 =$_C$ log2 =$_C$ ）。由于函数 G log2 G 是凸函数，根据詹森不等式，E/ log2 / > （E/) log2 E/。取 / 为 =0 或 =$_C$ =0 的概率相等，可以得出 =2 Clog2 =2C < =C $_{2^{=0}}$ log$_2$(=$_C$ = 0) + =2-0 log2（=0） ，等价地，=0 log2 =0 =$_C$ log2 =$_C$ > (=$_C$ =0) log2(=$_C$ =0) =$_C$。 因此 $\Delta_c > g_{C>C}$ (=$_C$ =0) log2(=$_C$ =0) 。 分别对（2）和（3）进行编码的成本大于或等于使用它们的联合分布进行编码的成本，并且至少存在 E = max{=$_C$ 等于 0，|Ω$_2$| 这些值的总成本必须至少为 E log☒ E。因此个不同的

目标的变化必须至少为 E log☒ E + Δ_c > E log☒ E gC>C E log☒ E > (1 - gC>C) |Ω☒| log☒ |Ω☒| 。

☒

5. 若“！☒（S0|D）” “! ☒（S|D）”，则倾向于进行拆分。C 中的列对 S 和 S0 的对数似然贡献相同，因此可以忽略不计。拆分出来的表 C 0 仅对 S0 的目标有贡献，且为！4=6C⌘（C 0）。这样就只剩下 S0 中新增的外键与 S 中列 C 的贡献之间的比较。在 S0 中，外键的分布是列 C 的经验联合分布。展开对数似然的差值可得

$$J_{MLE}(\mathcal{S}'|\mathcal{D}) - J_{MLE}(\mathcal{S}|\mathcal{D})$$
$$= n_t \sum_x \hat{p}_C(x) \left(\log \hat{p}_C(x) - \sum_{c\in C} \log \hat{p}_c(x) \right) - Length(t')$$
$$= n_t \sum_x \log \hat{p}_C(x) \left(\hat{p}_C(x) - \log \prod_{c\in C} \hat{p}_c(x) \right) - Length(t')$$
$$= KL\left(\hat{p}_C \middle\| \prod_{c\in C} \hat{p}_c \right) - Length(t').$$

参考文献

[1] Ziawasch Abedjan, Patrick Schulze, and Felix Naumann. 2014. DFD: E cient Functional Dependency Discovery. In CIKM 2014. 949–958.

[2] Periklis Andritsos, Renée Miller, and Panayiotis Tsaparas. 2004. Information-Theoretic Tools for Mining Database Structure from Large Data Sets.. In SIGMOD. 731–742.

[3] Anonymous. 2022. Schema Learning. https://anonymous.4open.science/r/schema_learning-EC93/README.md.

[4] Catriel Beeri and Philip A. Bernstein. 1979. Computational Problems Related to the Design of Normal Form Relational Schemas. ACM Trans. Database Syst. 4, 1 (March 1979), 30–59. https://doi.org/10.1145/320064.320066

[5] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. 1983. On the Desirability of Acyclic Database Schemes. J. ACM 30, 3 (July 1983), 479–513. https://doi.org/10.1145/2402.322389

[6] Philip A. Bernstein. 1976. Synthesizing Third Normal Form Relations from Functional Dependencies. ACM Trans. Database Syst. 1, 4 (Dec. 1976), 277–298. https://doi.org/10.1145/320493.320489

[7] Laure Berti-Équille, Hazar Harmouch, Felix Naumann, Noël Novelli, and Sara-vanan Thirumuruganathan. 2018. Discovery of Genuine Functional Dependencies from Relational Data with Missing Values. Proc. VLDB Endow. 11, 8 (April 2018), 880 892. https://doi.org/10.14778/3204028.3204032

[8] Philip Bille. 2005. A survey on tree edit distance and related problems. Theoretical computer science 337, 1-3 (2005), 217–239.

[9] C Chow and Cong Liu. 1968. Approximating discrete probability distributions with dependence trees. IEEE transactions on Information Theory 14, 3 (1968), 462–467.

[10] Jim Diederich and Jack Milton. 1988. New Methods and Fast Algorithms for Database Normalization. ACM Trans. Database Syst. 13, 3 (Sept. 1988), 339–365. https://doi.org/10.1145/44498.44499

[11] Mathias Drton and Marloes H Maathuis. 2017. Structure Learning in Graphical Modeling. Annual Review of Statistics and Its Application 4, 1 (2017), 365–393.

[12] Peter A. Flach and Iztok Savnik. 1999. Database dependency discovery: a machine learning approach. Ai Communications 12, 3 (1999), 139–160.

[13] Musicbrainz foundation. 2020. Musicbrainz. data retrieved from , https:// musicbrainz. org/.

[14] Yihan Gao and Aditya Parameswaran. 2016. Squish: Near-Optimal Compression for Archival of Relational Datasets. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (San Francisco, California, USA) (KDD ' 16). Association for Computing Machinery, New York, NY, USA, 1575–1584. https://doi.org/10.1145/2939672.2939867

[15] Bogdan Ghita, Diego G. Tomé, and Peter A. Boncz. 2020. White-box Compression: Learning and Exploiting Compact Table Representations. In CIDR 2020. http: //cidrdb. org/cidr2020/papers/p4-ghita-cidr20.pdf

[16] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. 1999. Tane: An E cient Algorithm for Discovering Functional and Approximate Dependencies. Comput. J. 42, 2 (1999), 100–111.

[17] Amir Ilkhechi, Andrew Crotty, Alex Galakatos, Yicong Mao, Grace Fan, Xiran Shi, and Ugur Cetintemel. 2020. DeepSqueeze: Deep Semantic Compression for Tabular Data. In SIGMOD 2020. Association for Computing Machinery, New York, NY, USA, 1733–1746.

[18] Ali Jalali, Christopher C Johnson, and Pradeep Ravikumar. 2011. On Learning Discrete Graphical Models using Greedy Methods. In NIPS.

[19] Lan Jiang and Felix Naumann. 2020. Holistic primary key and foreign key detection. Journal of Intelligent Information Systems 54 (06 2020), 1–23. https: //doi.org/10.1007/s10844-019-00562-z

[20] Batya Kenig, Pranay Mundra, Guna Prasaad, Babak Salimi, and Dan Suciu. 2020. Mining Approximate Acyclic Schemes from Relations. In SIGMOD 2020 (Portland, OR, USA). New York, NY, USA, 297–312. https://doi.org/10.1145/3318464.3380573

[21] Jyrki Kivinen and Heikki Mannila. 1995. Approximate inference of functional dependencies from relations. Theoretical Computer Science 149, 1 (1995), 129–149.

[22] Stéphane Lopes, Jean-Marc Petit, and Lot Lakhal. 2000. E cient Discovery of Functional Dependencies and Armstrong Relations. In EDBT 2000, Vol. 1777. 350–364.

[23] Panagiotis Mandros, Mario Boley, and Jilles Vreeken. 2017. Discovering Reliable Approximate Functional Dependencies. In KDD 2017. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3097983.3098062

[24] Panagiotis Mandros, Mario Boley, and Jilles Vreeken. 2018. Discovering reli-able dependencies from data: Hardness and improved algorithms. In 2018 IEEE international conference on data mining (ICDM). IEEE, 317–326.

[25] Jan Motl and Oliver Schulte. 2015. The CTU Prague Relational Learning Reposi-tory. CoRR (2015). http://arxiv.org/abs/1511.03086

[26] Noel Novelli and Rosine Cicchetti. 2001. FUN: An E cient Algorithm for Mining Functional and Embedded Dependencies. In ICDT ' 01 Proceedings of the 8th International Conference on Database Theory. 189–203.

[27] North Carolina Board of Elections. 2020. Voter registration. data retrieved from , https://www.ncsbe.gov/results-data/voter-registration-data.

[28] Thorsten Papenbrock, Tanja Bergmann, Moritz Finke, Jakob Zwiener, and Felix Naumann. 2015. Data Pro ling with Metanome. Proc. VLDB Endow. 8, 12 (Aug. 2015), 1860–1863. https://doi.org/10.14778/2824032.2824086

[29] Thorsten Papenbrock and Felix Naumann. 2016. A Hybrid Approach to Functional Dependency Discovery. In SIGMOD 2016 (SIGMOD ' 16). New York, NY, USA.

[30] Thorsten Papenbrock and Felix Naumann. 2017. Data-driven Schema Normaliza-tion. In EDBT 2017.

[31] D. S. Pavlichin, A. Ingber, and T. Weissman. 2017. Compressing Tabular Data via Pairwise Dependencies. In 2017 Data Compression Conference (DCC). 455–455. https://doi.org/10.1109/DCC.2017.82

[32] Mateusz Pawlik and Nikolaus Augsten. 2015. E cient computation of the tree edit distance. ACM Transactions on Database Systems (TODS) 40, 1 (2015), 1–40.

[33] Christian P Robert et al. 2007. The Bayesian choice: from decision-theoretic foundations to computational implementation. Vol. 2. Springer.

[34] Catharine Wyss, Chris Giannella, and Edward L. Robertson. 2001. FastFDs: A Heuristic-Driven, Depth-First Algorithm for Mining Functional Dependencies from Relation Instances - Extended Abstract. In DaWaK International Conference on Data Warehousing and Knowledge Discovery. 101–110.

[35] Hong Yao, H.J. Hamilton, and C.J. Butz. 2002. FD/spl I.bar/Mine: discovering functional dependencies in a database using equivalences. In 2002 IEEE International Conference on Data Mining. 729–732.

[36] Yunjia Zhang, Zhihan Guo, and Theodoros Rekatsinas. 2020. A Statistical Per-spective on Discovering Functional Dependencies in Noisy Data. In SIGMOD 2020.