Journal of Intelligent Information Systems (2020) 54:439–
461 https://doi.org/10.1007/s10844-019-00562-z

整体主键和外键检测

Lan Jiang[1] **·Felix Naumann**[1]

收稿日期：2019 年 2 月 28 日 / 修订日期：2019 年 5 月 20 日 / 接受
日期：2019 年 5 月 21 日 / 在线发表日期：2019 年 6 月 10 日

摘要

主键（PK）和外键（FK）是各种应用中关系模式的重要组成部分，例如查询优化和数据集成。然而，在许多情况下，这些约束条件是未知的或未被记录。手动检测它们既耗时又在大规模数据集中难以实现。我们研究了自动发现主键和外键的问题，并提出了一种同时检测两者的算法，即整体主键和外键检测（HoPF）。主键和外键分别是唯一列组合（UCC）和包含依赖关系（IND）的子集，对于这两者，已知存在高效的发现算法。通过使用评分函数，我们的方法能够从大量的有效 UCC 和 IND 中有效地提取出真正的主键和外键。我们采用了若干剪枝规则来加快处理速度。我们在三个基准数据集和两个真实世界数据集上评估了准确率和召回率。结果表明，我们的方法平均能够检索到 88% 的所有主键和 91% 的所有外键。我们将 HoPF 的性能与两个假设存在主键的基线方法进行了比较。

关键词 数据剖析应用程序 · 主键 · 外键 · 数据库管理

1 结构模式

主键（又称*键*）和*外键*是关系型数据库中最重要的两种约束，表明了数据库需要遵循的实体完整性和参照完整性。这两种约束在数据库中被广泛使用。在理想情况下，这些约束应由数据库设计者明确指定。然而，在许多情况下

Lan Jiang lan.jiang@hpi.de

Felix Naumann felix.naumann@hpi.de

[1] Hasso Plattner Institute, University of Potsdam, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany

在实际案例中，主键和外键要么不完整，要么缺失，这使得理解模式的结构变得困难。缺乏约束的原因多种多样。例如，对于被视为事实表且通常具有多列主键的表，模式设计者可能不会定义主键。此外，开发人员对于非数字且不会自动递增的列，往往不愿意使用主键约束。另一方面，出于效率方面的考虑，有时不会为外键定义约束：它们可能会影响数据库管理系统中插入、更新和删除记录的速度。一种常见的做法是在应用层控制参照完整性。在迁移数据库时，尤其是通过数据库转储和扁平文件进行迁移时，所有约束定义都可能丢失。

主键和外键的知识对于诸如数据清理、逆向工程、查询优化和数据集成等应用来说至关重要。尽管对于小型数据库，缺失的主键和外键仍可由领域专家手动标注，但对于规模较大的数据库模式，这样做极其耗时甚至不可行。虽然已经有一些针对外键检测的研究（Rostin 等人，2009 年；Zhang 等人，2010 年），但它们都假定主键的存在。通常，存储在关系数据库管理系统（RDBMS）平台中的数据都有主键，因为这些工具会明确要求用户为每个表指定主键。然而，对于从平面文件或网络来源编译的数据库，情况并非总是如此，因为此类约束信息需要存储在单独的文件中。根据我们的经验，这种伴随信息往往不存在。这种缺失促使我们也要检测主键，并设计一种方法来整体解决这两个相互依赖的问题。

主键和外键通常可以涵盖多个属性，本质上是*唯一列组合*（UCC）和*包含依赖关系*（IND）的特殊情况。幸运的是，先前的研究（Abedjan 等人，2015 年）已经提出了许多用于发现 UCC 和 IND 的算法，这使得这些元数据易于获取。然而，这些算法通常会生成大量的 UCC 和 IND，远远多于实际的主键和外键。因此，任务在于从虚假的 UCC 和 IND 中区分出真正的主键和外键。通过检查数据，可以找到一些直观的规则来区分键和非键，这使得自动检测键成为可能。根据外键的定义，它必须引用主键。因此，检测到的外键依赖于我们找到的主键。在本研究中，我们设计了一种算法，首先为模式发现主键，然后基于这些主键发现外键。反过来，预测的外键有助于去除一些错误预测的主键，从而提高整体结果的质量。第 3 节将更正式地介绍问题定义，不过通过图 1 我们已经给出了主键和外键检测难度的一个激励性示例。

对于一个表来说，其可能存在的众多唯一列组合都是该表主键的候选。例如，图 1 中的 *Trade* 表有两个唯一列组合，即 *T ID* 和 *T CA* ID。虽然也可以将 *T ID*、*T CA ID* 视为唯一列组合，但为简便起见，我们只考虑最小的唯一列组合（详情见第 3.2 节）。根据模式文档，这些唯一列组合中只有 *T ID* 是真正的主键，而另一个则是虚假的候选。在实际案例中，可能会有更多虚假的候选主键，这使得识别真正的主键颇具挑战性。

除了真正的外键之外，我们还会遇到许多虚假的包含依赖关系。例如，IND Trade.*TT ID* ⊆TradeType.TT *ID* 是一个真正的外键，而 IND Trade.*T ID* ⊆Company.CO *ID* 只是因为某些原因才成为外键候选。

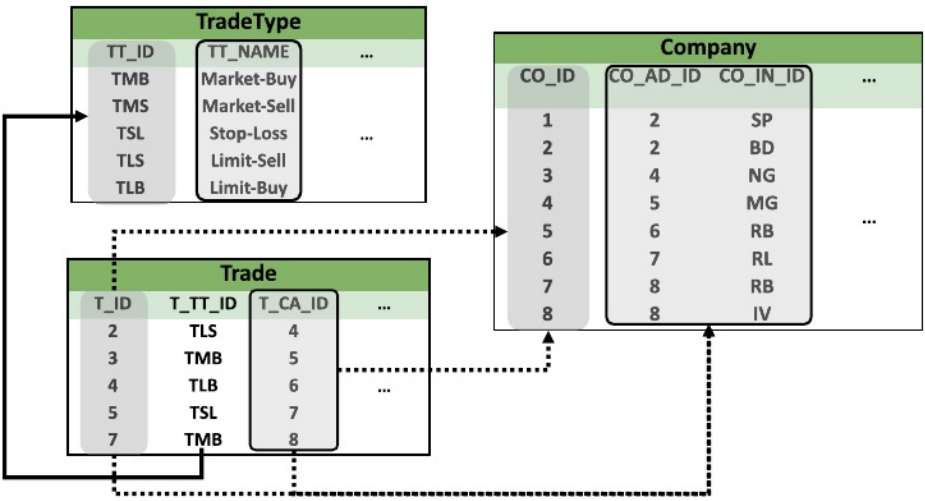


图 1 *TPC-E* 模式的部分表。为简洁起见，省略了一些列。真正的主键用未加框的灰色方框表示，而虚假的唯一组合键用加框的灰色方框表示。真正的外键用实线箭头表示，而虚假的索引用虚线箭头表示。

满足了包含关系。如图 1 所示，可能存在大量虚假的外键依赖，而真正的外键依赖则要少得多。

在其他情况下（图中未显示），两个列组合可能相互包含。例如，*TPC-E* 列 {Trade.T *ID}* 和 {Settlement.SE *T ID}* 分别是表 Trade 和 Settlement 的主键，并且它们恰好包含相同的值集。因此，INDs Trade.*T ID* ⊆Settlement.SE *T ID* 和 Settlement.*SE-T ID* ⊆Trade.T *ID* 都被视为外键候选，但只有后者才是真正的外键。

由于主键和外键发现的结果应由专家用户进行验证，因此我们在评估中同时考虑了准确率和召回率。然而，我们提出的算法更注重召回率，因为我们认为用户从少量结果中移除误报要比在庞大的候选集中发现漏报更容易。我们方法的贡献在于以下几点：

我们提出了首个联合主键/外键发现算法，不再假定主键已知且存在。

我们提出先进的剪枝规则，以过滤掉虚假的唯一列组合和包含依赖关系。

在五个不同数据集上进行的对比实验表明了我们整体算法的有效性。在不分配主键的情况下生成外键，也证明了主键缺失会降低外键检测的性能。

我们在第 2 节讨论相关工作，在第 3 节给出研究问题的正式定义。第 4 节列出了用于主键和外键检测算法的特征，第 5 节描述了主键和外键候选的剪枝策略。第 6 节解释了整体算法。实验结果在第 7 节展示，并与相关工作进行了比较。我们在第 8 节总结。

2 相关工作

本文提出的算法需要唯一列组合（UCC）和包含依赖（IND）作为输入。虽然发现这些内容并非本文重点，但我们简要总结了最新的技术，随后讨论了关于主键和外键发现的先前研究。

2.1 元数据发现

对于给定的数据集，高效地发现所有唯一列组合（UCC）和包含依赖关系（IND）是一项挑战，原因在于列组合的组合爆炸式增长。发现所有最小唯一列组合和所有最大 n 元包含依赖关系的问题属于 NP 完全问题（Lucchesi 和 Osborn 1978；Kantola 等人 1992）。幸运的是，已经有不少算法被设计出来用于发现唯一列组合和包含依赖关系（Abedjan 等人 2015），这些算法在实际应用中是高效的。然而，它们的结果规模可能相当庞大。即使在只有几十列的表中，发现数百甚至数千个唯一列组合和包含依赖关系的情况也并不罕见。挑战在于从如此庞大的候选集中找出真正的主键和外键。

2.2 关键约束发现

令人惊讶的是，针对主键检测的研究工作并不多。曾有人提出了一组简单的启发式特征，用于区分真正的主键和虚假的唯一列组合，以将关系分解为博伊斯 - 科德范式（Papenbrock 和 Naumann 2017）。作者为每个唯一列组合计算并累加其特征得分。由于没有人类专家来查看排序结果，因此将表中得分最高的唯一列组合假定为主键。

人们为发现外键做出了更多努力（Lopes 等人，2002 年；Memari 等人，2015 年；Rostin 等人，2009 年；Zhang 等人，2010 年；Chen 等人，2014 年；Marchi 等人，2009 年）。在此，我们简要介绍其中三项具有代表性的研究成果。与前面提到的主键发现思路类似，也可以为外键检测提出一些直观的特征。Rostin 等人提出了十个特征，用于机器学习方法从各种数据集中自动检测外键（Rostin 等人，2009 年）。然而，该方法仅能检测单列外键。作者列举了分类器出错的几种情况，包括传递外键和一对一关系。传递外键指的是这样一种情况，即被外键引用的主键本身也是一个外键。一对一关系在本文中被表述为“PK ⊆ PK”，并将在下文进行解释。他们的基于机器学习的方法无法解决这两种情况。

假设外键的数据应当很好地代表其所引用的主键列的一个样本，一种先进的方法引入了*随机性*度量来发现单列和多列外键（Zhang 等人，2010）。作者使用了“地球移动距离”（EMD）来衡量外键候选的左、右两侧数据分布的相似性：它是将一个分布转换为另一个分布时在分布内各桶之间移动值计数的最小成本（Rubner 等人，1998）。通过为左、右两侧选择相同数量的固定桶，并计算每个桶中对应值的数量来创建数据分布。数据分布越接近，距离就越小。然后根据其“地球移动距离”对所有外键候选进行排序，并基于此报告性能。

结果中的前 X%。我们通过实验表明，在大多数情况下，我们的方法在准确率和召回率方面都优于这项工作。

陈等人提出将启发式特征与不同的剪枝规则相结合来检测外键，这与我们的算法最为相似（陈等人，2014 年）。然而，他们假设每对表只能包含一个外键，这对于现实场景来说是一个过于严格的限制。

我们注意到，上述*所有*方法都假定主键已知，并基于此构建其启发式算法。我们摒弃了这种先验知识的假设，提出了一种更适合许多实际场景的方法。

3 问题设定

在介绍问题之前，我们先简要定义本文中使用的术语。我们将关系模式表示为 R，它可以包含多个关系。每个关系表示为 *R*，是一个非空的属性命名集合。属性表示为 $A \in$ R，其域为 *dom(A)*。每个关系包含一组元组；*R* 的实例表示为 r。*r* 中的一个元组表示为 t。基于上述定义，元组 *t* 可以表示为 $_{A \in R}\, t[A]$，对于每个属性 *A*，其值 *t[A]* ∈ dom(A)。我们使用 *t[X]* 表示 *t* 在 $X \subseteq$ R 的值上的投影。设 **U** 和 **I** 分别是从相应的发现算法中提取的唯一列组合集和包含依赖集。

3.1 依赖关系的类型

描述表的特征或表与表之间关系的依赖项有很多种。由于唯一列组合和包含依赖项分别是主键和外键的前提条件，所以我们仅使用这两种依赖项。我们将给出这些依赖项的定义以及它们与主键和外键之间的关系。

定义 1（唯一**列组合**）给定关系 *R* 及其实例 *r*，*唯一列组合*（UCC）是 R 的属性集 *X*，其投影在 r 上仅包含唯一且非空值的条目，即对于所有 t_i，$t_j \in$r，$i \neq j$：[X] $t_i \neq$ [X] t_j，并且对于所有 t ∈R，*A* ∈X：*t[A]* ≠ ⊥。

请注意，列组合可能包含一个或多个列，因为主键可以由一个或多个属性组成。实际上，我们观察到商业模式中主键定义包含多达 16 个属性（Faust 等人，2014 年）。当一个 UCC 的子集不存在有效的 UCC 时，该 UCC 即为最小的。原则上，每个关系 *R* 应该有一个且仅有一个主键。尽管在实际中，某些表可能无法定义主键，但我们假定每个表都不含重复项，因此至少存在一个 UCC。

模式中的每个表都包含若干个唯一性组合键（UCC），而每组中只有一个才是真正的主键。从每组中选取一个，我们可以构成一个代表关系 R 的主键的唯一性组合键列表，并用 *PKR 表示*。

定义 2（包含**依赖**）给定两个关系 R_i、$R_j \in$R，$i = j$，*包含依赖*（IND），记作 R_i [X] $\subseteq R_j$ [Y]，表示在列组合 *X* 中的所有值项也包含在列组合 *Y 中*，即对于关系实例 r_i 和 r_j，$\forall t_i$ [X] $\in r_i$，$\exists t_j$ [Y] $\in r_j$：t_i [X] = t_j [Y]。

我们将依赖部分 R_i [X] 称为左部（LHS），将被引用部分 R_j [Y] 称为右部（RHS）。请注意，IND 的定义意味着 |X| = |Y|。当 |X| = |Y| = 1 时，我们称该 IND 为*一元的*，否则为 *n* 元的。外键必须是一个 IND，因为根据定义，外键左部出现的每个值都必须包含在其右部的值集中。

3.2 问题定义

我们任务的正式问题陈述为：

> **主键/外键检测问题：**给定具有模式 R 的数据库、其最小唯一候选键集 **U** 以及其函数依赖集 **I**，找出主键集和外键集，分别记为 **P** 和 **F**，其中 **P** ⊆ U 且 **F** ⊆ I。

主键和外键之间的依赖关系显而易见，因为外键的右侧必须是主键。尽管每个表可能有多个候选键，但我们假定其中只有一个才是真正的主键。在这项工作中，我们仅使用最小唯一列组合（UCC），原因有二：1）所提出的主键特征（详见第 4 节）总是倾向于选择最小 UCC 而非其非最小的超集；2）完整的 UCC 集合（包括最小和非最小 UCC）可能比最小 UCC 集合多出指数级的 UCC，因为包含某个 UCC 的每一列组合也都是有效的 UCC。不过，我们也意识到这一选择存在一个缺点：在对小型关系进行主键预测时，真正的多列主键不会被添加到主键候选集中，因为该主键集的一个子集已经是最小 UCC。我们将在第 7 节进一步讨论这个问题。

为了防止过多地提示错误的外键，我们的算法会在模式中的所有关系都已连接或不再有候选外键时停止外键检测。模式中主键的不同自动选择可能会导致外键选择的不同。为了全面解决这个问题，我们在基于分数的方法中采用了剪枝规则，详情将在以下章节中进行说明。

为简便起见，我们假设输入的仅是精确的唯一条件组合（UCC）和内部依赖（IND），但在实际情况中数据可能存在错误。近似检测算法可解决这一问题，发现可能存在违反情况的 UCC 和 IND（Ilyas 等人，2004 年）。我们的方法对这类输入变化不敏感，仍能为键和外键提供良好的建议。

4 主键和外键的特性

先前的研究已经提出了一些用于识别主键（Papenbrock 和 Naumann 2017）或外键（Rostin 等人 2009；Zhang 等人 2010）的有用特征。我们采用了其中的一些特征，并定义了一些新的特征，以对主键和外键的每个候选进行评分。在本节中，我们将简要介绍每个特征，然后在下一节中使用它们。

4.1 主要特点

为了区分真正的主键和虚假的唯一候选键（UCC），已有研究探索了几个有用的启发式特征（Papenbrock 和 Naumann 2017），包括 UCC 的基数、平均值长度以及其属性在模式中的位置。作者已证明这些特征对于其规范化应用中的主键发现是有效的。

我们在这项任务中复用了这三个特征的评分函数。根据我们的观察，在很多情况下，主键列的表头遵循的模式与虚假候选列的表头不同。因此，我们引入了一个额外的基于名称的特征。

基数 我们更容易理解和维护属性较少的主键。在此，唯一组合键（UCC）所涉及的列越多，其成为真正主键的可能性就越低。我们将基数得分定义为 $\frac{1}{|X|}$ 其中 X 是 UCC 的列集。

用作主键的列的值长度通常较短，因为它们通常是无语义含义的标识符。此外，对主键的长值进行索引可能会降低效率。此功能所使用的评分函数为 $\frac{1}{\max(1,|\max(X)|-n}$ 其中 $|\max(X)|$ 是每个值中最长值的长度平均值

在 X 中的列，n 表示用于惩罚长值的参数。可根据数据集进行调整。在我们的实验中，我们选择 $n = 8$，这反映了主键数据类型的典型选择。对于多列唯一组合键，我们计算每列的得分，并使用平均值作为其总体得分。虽然在得分函数中使用常量会影响通用性，但实验结果表明它非常有用。

在原则上，关系中的属性是无序的，但在实际定义模式时，属性的位置存在隐含的顺序。在大多数情况下，主键会出现在列集的最前面。对于网页表格也有类似的观察结果（Venetis 等人，2011 年）。此外，对于多列主键，我们期望在主键列之间没有（或只有很少）非主键列。列位置得分计算公式为 $12\,(\,|\text{left（X)}|+11 + |\text{between(X)}|+11\,)$，其中 *left(X)* 和 *between(X)* 分别表示数量。

分别是 X 的第一列左侧的列数以及 X 的第一列和最后一列之间的非键列数。

名称后缀 我们注意到在所有用于实验的数据集中，主键列通常通过其列名后缀来标识，例如“id”和“key”。在此，我们选择“id”、“key”、“nr”和“no”作为我们的后缀集。显然，此列表可以扩展，例如，可以包含其他语言的模式。我们应用的评分函数为 $|\text{suffix(X)}| / |X|$，其中 $|\text{suffix(X)}|$ 计算

在 UCC 中名称包含上述任一后缀的列的数量。我们将上述特征分数的平均值表示为主键候选分数。由于我们不能期望所有模式的标签都如此配合，因此我们也进行了关闭此特征的实验。

4.2 主要特点

Rostin 等人提出了十种启发式特征来在 IND 中发现外键（Rostin 等人，2009 年）。Zhang 等人提出了一种数据值分布的*随机性*特征，以涵盖除列名之外的多种此类特征（Zhang 等人，2010 年）。我们验证了这种覆盖的有效性，并将数据分布也作为衡量标准。然而，在实践中，由于要为 n 元外键候选者构建多维直方图，发现外键的分析时间相当长。因此，我们简单地将 n 元外键候选者的每个维度视为单列候选者，并对分数取平均值。列名相似性也起着重要作用，这并非数据分布特征所涵盖的。因此，我们采用两种特征来发现外键，即*列名*和*数据分布*。

列名 在很多情况下，为了更好地理解和维护数据库，数据库设计者不会给相关的**列**随意命名。对于一个 IND *R.A* ⊆S.B，如果 *R* 和 A 的标签与 *B* 和 *S* 的标签相似，则更有可能表示一个真正的外键。我们测试了几种字符串相似性函数，发现最初为解决记录之间的模糊匹配而提出的模糊相似性函数（Chaudhuri 等人，2003 年）最适合我们的任务。使用此度量计算相似度时，首先通过分隔符（如大写字母和“”）对每个字符串进行分词。对于每个外键候选，我们将 LHS 和 RHS 的表名和列名的标记集组合起来。例如，对于一个 IND Trade.*T S SYMB* ⊆LastTrade.LT S SYMB，LHS 和 RHS 的列名分别被转换为 T_L = {Trade，T， S， SYMB} 和 T_R = {Last, Trade， LT， S, SYMB}。在计算 LHS 和 RHS 的相似度时，我们将 T_R 中的每个标记映射到 T_L 中最相似的未映射标记，通过莱文斯坦距离计算。*L* ⊆R 的相似度根据以下公式计算（Chaudhuri 等人，2003 年）：

$$sim(L, R) = \frac{\sum_{j=1}^{n}(sim_j \times ln(\frac{1}{f_j}))}{\sum_{j=1}^{n} ln\frac{1}{f_j}}$$

其中 sim_j 是 T_R 中每个标记与 T_L 中与其最相似的标记之间的标记相似度。如果 T_R 中的标记少于 T_L 中的标记，那么 T_R 中的一些标记无法找到映射。对于每个这样的标记 *j*，我们将 simj 设为零。*f* 表示每个标记的频率，*ln*(f1j) 是 T_R 中每个标记的权重，用于衡量标记的稀有程度。直观地说，稀有标记对于指示两个字符串之间的相似性更有用，因此应赋予更高的权重。相似度是不对称的，这意味着对于两个字符串 *L* 和 *R*，L ⊆ R 的相似度与 *R* ⊆ L 的相似度不同。这有助于在两个列集相互包含且其数据分布得分相同时识别真正的外键。

数据分布 参与列的数据分布是区分外键和虚假 IND 的良好指标。在 Zhang 等人（2010 年）的研究中，作者假设从属列的值应从所引用列的值中均匀采样，并提出使用“地球移动距离”作为成本度量。我们验证了这一假设的有效性，但注意到构建此度量的时间开销较大。因此，我们提出了一种更简单但（如实验所示）有效的直方图差异来表示成本：对于 IND R_i [X] ⊆R_j [Y]，我们根据 [Y] 中每列 Y_i 的值范围创建一组桶，并将每个值放入相应的桶中。默认情况下，我们选择二十个桶，第 7.2 节的实验表明这是一个不错的选择。这些桶形成一个直方图，记为 *Hist*(Y_i，对于 *X* 中的每列 X_i，我们将它的值放入为 Y_i 创建的桶中。整体数据分布得分是每个维度直方图差异得分的平均值。虽然还有一些其他的选择，例如 L_1 范数和直方图相交，但我们使用巴氏系数（Bhattacharyya 1943）——一种用于衡量两个直方图之间相似性的公认良好方法。

5 修剪主键和外键候选者

在搜索空间中枚举并评分所有 UCC 和 IND 的组合是发现主键和外键的最朴素方法。由于其耗时极长，这种方法实际上并不可行。

潜在的候选数量呈指数级增长。考虑一个仅有 20 个表的模式，每个表仅包含两个唯一列组合（UCC），整个数据库仅包含 50 个索引依赖（IND）。那么在搜索空间中就有 2^20 种主键候选组合和 2^20 种外键候选组合。而在实际应用中，表包含的唯一列组合要多得多，典型的数据库包含数千个索引依赖。

外键被定义为引用主键。这促使我们将主键和外键的发现作为一个整体来进行。具体来说，从搜索空间中选择一组唯一的列组合（UCCs），将其视为预测的主键。然后，从右值（RHS）属于这些主键的函数依赖（INDs）中确定外键。将此选择的主键得分与外键得分相加。在枚举所有 UCCs 组合及其对应的 INDs 之后，选择总分最高的 UCCs 和 INDs 作为输出的主键和外键。

为避免对每个 UCC 和 IND 进行检查，本章建议了一些用于主键和外键发现的筛选技术，以帮助算法跳过不必要的检查。这些技术包括外键候选预筛选、$PK \subseteq$ PK 筛选、主键候选修剪和外键候选修剪。我们在本章的其余部分将详细讨论每一种技术。

5.1 FK 候选人预筛选

索引数量通常会随着表和列的数量呈二次方增长（Tschirschnitz 等人，2017 年）。然而，并非所有索引都是外键的良好候选者。预筛选步骤旨在仅保留一小部分合适的索引作为外键候选。我们使用以下规则来优化原始索引集：

外键只能引用主键，而主键的定义是唯一的候选码且不含空值。因此，我们剔除所有右侧不是唯一候选码（即不是主键候选）的函数依赖。

全空值列组合 我们注意到存在只有空值的列，这种情况在实际数据集中尤为常见。原则上，这些列可以被视为包含在任何其他列中。然而，对于派生外键而言，它们并无用处，因此我们将其忽略。

表 1 展示了预过滤前后的 IND 数量（我们在第 7.1 节将更详细地介绍这些数据集）。

5.2 $PK \subseteq$ PK 筛选

如果一个 IND 的左值是右值的一个连续子序列，这看起来可能是一个不错的外键候选。然而，在很多情况下，实际上并非如此。

表 1 FK 候选预筛选前后的 INDs

数据集	之前	之后
TPC-H	90	33
TPC-E	511	175
广告作品	19,602	2,047
SCOP	6,450	2,062
MusicBrainz	236,151	28,722

对于这两个表的自动递增的一元主键。给定一个包含关系 $A \subseteq B$，如果 A 中的有序值是 B 中有序值的一个连续子序列，我们就移除它——在这种情况下，A 很可能是独立的键，将其包含在 B 中是多余的。

例如，在 TPC-H 中，所有表都有这样的自动递增整数主键。*REGION* 表仅包含五个元组，其主键值为 1 至 5。另一方面，*NATION* 表包含 24 个元组，其主键值为 1 至 24。因此，*REGION.REGIONKEY* ⊆ NATION.NATIONKEY 似乎是一个不错的外键候选。然而，这样的 *PK* ⊆ *PK* IND 是没有意义的，因此应该被过滤掉。

先前的研究未能很好地处理预测外键中存在 *PK* ⊆PK 候选的情况（Rostin 等人，2009 年；Chen 等人，2014 年）。鉴于大多数表都包含自动递增的整数主键，Zhang 等人（2010 年）的作者提出了一种在左表和右表之间进行连续前缀或后缀检查的简单方法来检测外键。而在我们的方法中，我们放宽了这一限制，考虑任何连续子序列。

5.3 主键候选列修剪

每张表可能拥有多个 UCC。为了获得最佳结果，我们需要考虑所有有效的 UCC 组合，即每张表所属的 UCC 的笛卡尔积。实际上，即使我们限制每张表仅考虑其中一个 UCC 作为主键，搜索空间仍然相当大。例如，从表 2 可以看出，对于 TPC-H 模式，有超过 7600 万个不同的 *PKR* 候选组合，记为 PKcc，即主键组合候选。

为了减少候选对象的数量，我们根据第 4.1 节中提到的主要关键特征对每个 UCC 进行评分。我们在每个表内按得分从高到低对它们进行排序，目的是剔除质量较差的候选对象。

图 2 展示了 TPC-E 模式中表 *Trade* 的每个主键候选的得分，作为示例。x 轴表示候选得分的排名。我们用 S_i 表示第 i 个最佳候选的得分。相邻候选*得分的差*值标记为 $SD_i = S_i - -S_{i+1}$。与连续曲线的*拐点*概念类似，我们为离散情况定义了一个悬崖：

定义 3（断崖） 给定表 *S 的主键候选者的排序分数列表* $S = \{S_1, S_2, ..., S_n\}$ 以及*对应的* SD *列表* $SD = \{SD_1, SD_2, ..., SD_{n-1}\}$，断崖是指相*邻候选者* Si 和 S*i*+1 中 SD 分数最大的一对。

表 2 剪枝*前后*具有主键候选组合的评估数据集 下降

数据集	桌子	PKcc 之前	繁荣正义党U	PKs L	PKcc 之后
TPC-H	8	7.67×10^{7}	8	0	1
TPC-E（事务处理性能委员会基准测试 E 型）	33	9.03×10^{13}	31	1	768
AdWork	27	6.23×10^{21}	27	0	256
SCOP	42	7.25×10^{11}	42	0	2
MusicBrainz	124	3.73×10^{25}	122	2	576

PKs U 和 *L* 分别表示落入上部或下部的真实主键的数量。因此，PKs L 是表的数量与 PKs U 的差值。

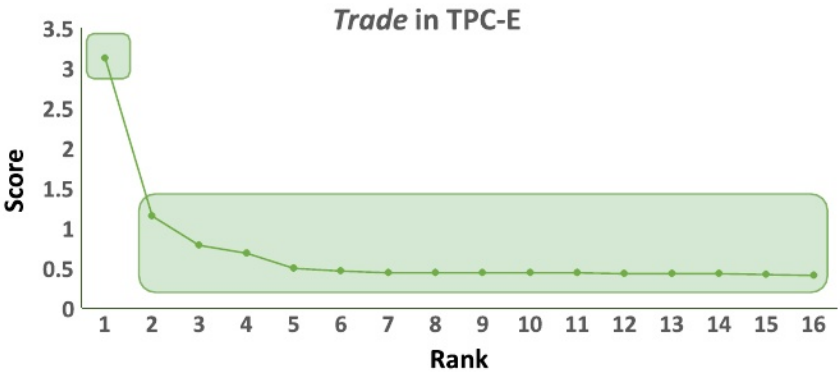


图 2 为 *TPC-E* 中 *Trade* 表的主键候选列的得分。第一个候选列和第二个候选列之间出现了*断崖*式下降。

每张表的候选键被分为两个子组，即“*上部*”，包含悬崖（S_1，...，S_i）之前的全部候选键，以及“*下部*”，包含其余的候选键（Si+1，...,Sn）。例如,*对*于“交易”表，悬崖*是由*于其 SD 分数最大而形成的前两个主键候选键的组合。在这种情况下，上部仅包含排名第一的候选键，而下部包含其余的 15 个候选键。

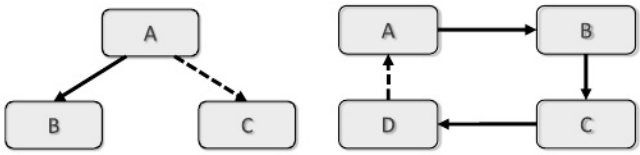
我们计算了所有数据集中的每个表的上限和下限，以查看真正的主键落入哪个部分。表 2 展示了结果。例如，*TPC-E* 中的 32 个主键中有 31 个落入上限，这意味着剪枝仅丢失了极少数的真正主键。表 2 中的统计数据也支持了其他数据集的这一观察结果。为了减少主键识别(*PKR*)的搜索空间，我们简单地将下限排除在主键竞争之外，从而大幅缩小了搜索空间。此步骤在算法 2 中以 *cliffPrune*( ) 函数实现。例如，SCOP 的搜索空间仅缩减为两个主键组合。另一方面，只有 TPC-E 和 MusicBrainz 丢失了主键。我们验证了使用 cliff 过滤掉劣质候选主键的有效性。通过使用这一概念，无需设置参数来控制过滤掉的候选主键数量。

5.4 外键候选列剪枝

用于生成外键候选的 IND 发现算法仅考虑列之间的值包含关系。然而，在从候选外键中查找真正的外键时，会出现两种类型的冲突。我们的方法需要在每次预测外键时验证不会产生冲突。我们在图 3 中解释了所采用的策略并进行了可视化。

外键的唯一性 每个外键在一个模式中只能引用一个主键，而一个 IND 的左部值可能包含在多个不同的右部中。如果这些 IND 中有一个是真正的外键（如图 3a 中的实线箭头所示），那么很明显其他所有都是虚假的（如图 3a 中的虚线箭头所示）。

非循环引用 如果模式中存在循环引用，则所有涉及的列组合都包含相同的值，我们认为这在语义上没有意义。因此，在模式图中，我们不会预测会导致循环引用的外键。例如，图 3b 中的虚线箭头引入了循环引用，应予以排除。



(a) Only one referenced RHS (b) Non-cyclic reference.

图 3 虚线表示的 IND 是无效的外键

如果其他三个实线箭头已被归类为外键，则不应将其预测为外键。

模式连接性 在完成上述处理和筛选之后，剩余的外键候选数量大幅减少。我们的算法会对每个候选进行评分，并选择一组优质的外键。为避免指定分数阈值，我们假设模式图实际上是一棵生成树。所有表的连接性是已找到大多数真实外键的良好指标，如图 5 中的召回率所示。为此，HoPF 每次预测新的外键时都会检查模式连接性是否已满足。

6 整体算法 HoPF

图 4 展示了我们所提出的 HoPF（整体*主键和外键检测*）*算法的整体流程*。它首先优化 UCC（唯一组合键）和 IND（函数依赖）的集合，从而减少

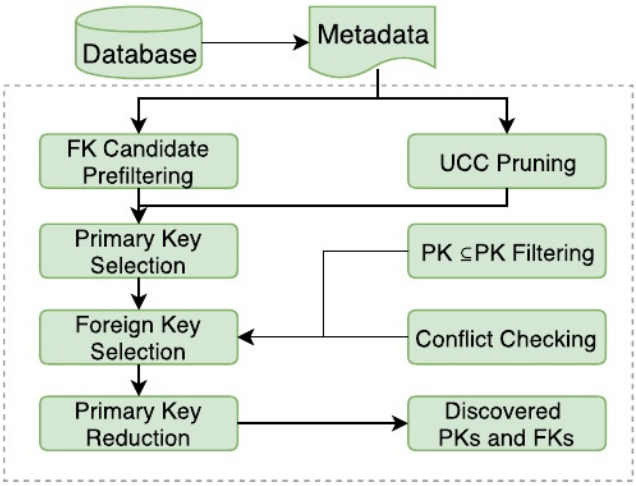


图 4 HoPF 算法概述

在下一步中需要处理的主键和外键候选的数量。对于所有幸存的候选，HoPF 枚举所有主键候选集及其对应的 IND，并根据第 4 节中提出的特征对它们进行评分，从而选择出所提议的真实主键和外键。在此过程中会移除 *PK* ⊆PK 候选。冲突检查被整合到外键检测过程中，以进一步移除虚假的 IND。在生成最终结果之前，会执行主键简化步骤来简化预测的主键，并修正相应的外键。请注意，所使用的候选修剪技术在第 5 章中有描述。

从唯一组合键（UCC）和索引（IND）整体确定主键和外键的通用算法如算法 1 所示，该算法又调用了分别展示的多种方法。

在算法 1 的第 1.2 行，即第 2 行，生成主键组合的搜索空间并将其保存在 *PKcc*（主键组合候选）中。从第 1.3 行开始的循环依次计算这些主键组合及其对应外键的得分，确定得分最高的组合作为预测的主键集和外键集。在第 1.4 行，仅保留引用预测主键的外键候选在 *FKCands* 中。在第 1.5 行，将外键的最小数量限制为 |PKc| - 1，否则肯定会违反模式连接性规则。在第 1.6 行，算法根据检测到的主键 *PKc* 搜索 IND 以形成外键。这里，*D* 表示被丢弃的包含依赖关系，稍后在第 1.7 行用于恢复一些外键。第 1.8 至 1.12 行只是选择主键及其对应的外键作为最终预测。

Algorithm 1 Holistic PK/FK detection (HoPF).

Input: UCC set ***U***, IND set ***I***
Output: predicted primary keys ***P***, predicted foreign keys ***F***

```
1  P = {}, F = {}, Q = 0;
2  PKcc = GetPKCombinations(U);
3  forall PKc of PKcc do
4      FKCands = I \ ¯runeCands(I,PKc);
5      if |FKCands| ≥ |PKc| − 1 then
6          (FKCands, D) = GetFKCandidates(FKCands);
7          (PKc, FKs) = PKReduce(PKc,FKCands,D);
8          score = Score(PKc) + Score(FKs);
9          if Q < score then
10             P = PKc;
11             F = FKs;
12             Q = score;
13 Return (P, F)
```

算法 2 展示了查找所有合适主键组合候选的步骤。它首先按所属表对唯一候选键（UCC）进行分组，并使用第 5.3 节所述的悬崖技术过滤掉较差的主键候选。为了组成一个主键组合，从每个表的剩余唯一候选键集合中选择一个候选。

Algorithm 2 GetPKCombinations().

Input: UCC set ***U***, relations ***R***
Output: primary key combinations ***PKcc***
1 $PKcc = \{\}$;
2 **forall** T_i *of* R **do**
3 $g_i = \{ \mid \in \wedge = \}$;
4 cli g_i
5 **forall** $\in g \times \ldots \times g_{|R|}$ **do**
6 $Pk = PKcc \cup \{ \}$
7 **return** *PKcc*

给定主键生成外键的过程如算法 3 所示。在 *I* 中的所有外键候选者按其得分进行排序。在第 3.3 行，我们初始化一个图来表示模式中的表以及它们之间的预测外键。每个节点代表一张表。如果两张表之间存在外键连接，则在对应的两个节点之间添加一条边。从第 3.4 行到第 3.15 行，根据得分从高到低依次将合格的 IND 作为预测的外键贪婪地添加到图中。每次预测一个新的外键时，算法都会检查模式的连通性，如果图已连通则停止。在第 3.6 行检查循环引用冲突，随后进行 $PK \subseteq PK$ 检查。未能通过 PK ⊆ *PK* 检查的候选者被添加到丢弃集 *D* 中，并用于主键缩减阶段。一旦预测了一个外键，所有具有相同左部的其他候选者都将被排除（第 3.14 行和第 3.15 行）。

Algorithm 3 GetFKCCandidates().

Input: inclusion dependencies ***I***
Output: foreign key candidates ***FKCands***, discard set ***D***
1 $FKCands = \{\}$; $D = \{\}$; $maxFKsScore = 0$;
2 $I' = SortByScoreDescending(I)$;
3 G(*I'*) = G(N=T,E={});
4 **while** *I' is not empty* **do**
5 $FirstElement(I')$;
6 *nce* **then**
7 $\subseteq$ **then**
8 $\cup \{FKCand\}$;
9 $\cup$
10 $G( )$
11
12
13 $\cup \{ \}$
14 $^{+} = \{ \mid LHS(Cand') = LHS(FKCand) \wedge FKCand' \subseteq I'\}$;
15 $= \setminus ^{+}$
16 **return** $(FKCands, D)$

原则上，每个表都应定义主键以保持实体完整性。但在实际操作中，主键并非总是被定义，尤其是对于所谓的关联表，它们代表 $m:n$ 关系。尽管索引主键有助于更高效地查询，但在频繁的数据修改情况下，它反而会成为一种负担，导致模式设计上的问题。

我们建议设计人员避免定义主键。我们注意到，在我们的实验中使用的两个最大的数据集，即 *SCOP* 和 *Musicbrainz*，都包含一些没有定义主键的连接表。由于我们仅使用最小的唯一候选键作为输入来生成主键候选集，如果连接表的真正多列主键的子集已经是唯一候选键，那么该真正主键可能不会包含在候选集中。如果我们的算法将这种表标记为主键，那肯定是误报。对于其左部是这种误报主键的真正外键，可能会因上述 $PK \subseteq PK$ 过滤而被标记为虚假的 IND。

一种避免这种真实外键丢失的简单方法是使用完整的唯一组合键（UCC）集作为输入来生成主键。然而，我们发现这种方法有两个缺点。首先，完整的 UCC 集包含的 UCC 数量比其最小集的对应数量呈指数级增长，因为每个包含 UCC 的列组合本身也是一个有效的 UCC。其次，为 HoPF 选择的主键特征并不倾向于非最小 UCC 而是倾向于最小 UCC。为了提高外键的召回率，我们提出将主键缩减作为后处理步骤，如算法 4 所示。我们观察到，通过移除这些不恰当的主键，我们可以重新发现一些在迭代外键候选时被丢弃的真实外键。对于每个预测的主键，我们将其从 *PKC* 中移除，并更新相应的外键候选。我们假设，如果整体得分在预测主键不存在时上升，则该预测主键不是真实的（应被移除）。这一过程因此带来了两个好处。一方面，一些误报的主键最终被移除，而之前因引用这些主键而被丢弃的 $PK \subseteq$ PK 候选也会被恢复，因为它们的左部已不再是主键。另一方面，那些错误地引用了这些已被移除的主键的外键将不再被视为外键。

Algorithm 4 PKReduce().

Input: primary key combination ***PKC***, predicted foreign keys ***FKCands***, discarded foreign key candidates ***D***
Output: updated primary keys ***PKC***, updated foreign keys ***FKs***

```
1  FKs={};
2  score = score_PKC + score_FKCands;
3  for each PK in PKC do
4      PKC' = PKC \ PK;
5      FKCands' = UpdateFKCands(PKC',FKCands, D);
6      if score < score_PKC' + score_FKCands' then
7          PKC = PKC';
8          FKs = FKCands';
9          score = score_PKC + score_FKs;
10     PKC = PKC \ P;
11 return PKC, FKs
```

7 实验与分析

我们接下来评估所提出的 HoPF 算法的有效性。在介绍实验设置之后，我们首先展示 HoPF 在各种数据集上所达到的准确率和召回率，然后分析三种不同类型的错误，即主键错误、左侧为空列以及 $PK \subseteq PK$ 错误。接下来，我们进一步探究并报告

表 3 数据集及其统计信息

名字	桌子	柱子	UCCs	对决	INDs	FKs
TPC-H	8	61	435	8	90	8
TPC-E（事务处理性能评估基准）		185	167	32	411	45
AdvWorks	27	321	1,434	27	4,300	45
SCOP	65	282	120	42	5,244	90
MusicBrainz	124	682	252	124	236,151	168

不同大小的桶用于数据分布特征对 F-1 分数的影响。之后，我们探讨了不分配主键时外键发现的性能，表明检测主键的必要性。最后，我们通过 HoPF 与最先进的算法（Zhang 等人，2010 年）的性能比较来结束本节。

7.1 实验装置

由于 HoPF 将 UCC 和 IND 作为输入，我们假定手头的数据集已经过分析，从而这些依赖关系以及基本统计信息都是可用的。鉴于有许多高效的分析算法（Abedjan 等人，2015 年），我们可以轻松获取这些依赖关系，因此这一假设是合理的。我们使用 *Metanome*（*www.metanome.de*）提取了所有这些元数据，这是一个数据分析平台，为不同的分析算法提供了广泛的接口（Papenbrock 等人，2015 年）。

我们在多个数据集上进行了实验，包括两个基准数据集 *TPC-H* 和 *TPC-E*[1]、一个生成的数据集 *AdventureWorksDW*[1] 以及两个真实世界的数据集 *SCOP*[3] 和 *MusicBrainz*[4]。所有数据集都包含完整的主键和外键定义，我们将其作为准确率和召回率评估的黄金标准。此外，所有数据集都包含每列的表头名称。数据集的详细信息见表 3。由于我们的算法是数据驱动的，我们移除了仅在 *MusicBrainz 中出现的空表,将其表的数量从 206 个减少*到 124 个。TPC-H 和 *TPC-E* 都有*几*个参数来控制其规模。为了便于与现有工作进行比较，我们将参数设置为与 Zhang 等人（2010 年）相同。

包含依赖的数量既包括单元的也包括多元的。HoPF 分别尝试从这些依赖中发现单列和多列的外键。在仔细检查数据集后，我们注意到尽管 *SCOP* 的模式包含 65 个表，但只有 42 个真正的主键被定义。我们观察到所有未定义主键的表都是连接表。我们尝试通过采用第 7.2 节中描述的主键缩减方法将这些表从结果中移除。

7.2 精确率和召回率分析

我们在每个数据集的整个模式上执行了 HoPF 算法。该算法以唯一组合键（UCC）、完整性依赖（IND）和基本统计信息作为输入，并分别输出 UCC/IND 的子集作为预测结果。

[1] http://www.tpc.org
[2] https://github.com/Microsoft/sql-server-samples/releases/tag/adventureworks
[3] http://scop.berkeley.edu
[4] https://musicbrainz.org

对于每个数据集，我们分别将输出集与该数据集的黄金标准进行比较，并报告准确率和召回率，即预测出的正确外键数量除以预测出的外键数量，以及预测出的正确外键数量除以黄金标准中的外键数量。图 5 展示了每个数据集中前 X 个预测外键的准确率和召回率。从图中可以看出，召回率逐渐上升，而准确率没有明显下降。也就是说，对于每个数据集，真正的外键通常出现在预测列表的顶部，这意味着它们的得分高于虚假的外键候选。当检查列表底部更多的预测条目时，我们看到的外键越来越少，而虚假的 IND 则越来越多。这证明了我们为 HoPF 选择的外键特征的有效性。

表 5 列出了 HoPF 发现的主键和外键的数量。它成功地为所有数据集发现了大多数真实的主键。在检查预测的主键集合后，我们发现错误出现有两个原因。首先，主键特征未能很好地涵盖那些错误的情况。例如，该算法未能找到 *TPC-E* 中 *Financial* 表的真正主键，因为该表包含一个三列主键和几个虚假的一元 UCC。这与我们的假设相悖，即主键应具有较小的基数。此外，真实主键列的名称不符合我们采用的后缀名称规则。这两个因素导致未能预测此关系的正确主键。然而，这种错误只占很小一部分。我们没有发现其他具有相同原因的失败情况，这表明我们用于主键发现的特征总体上是有效的。更多的错误源于主键的缩减步骤。我们将在下面讨论此策略对主键和外键发现性能的影响。我们还研究了由 HoPF 引起的预测外键的实际误报和漏报案例，发现它们可归为以下三类:

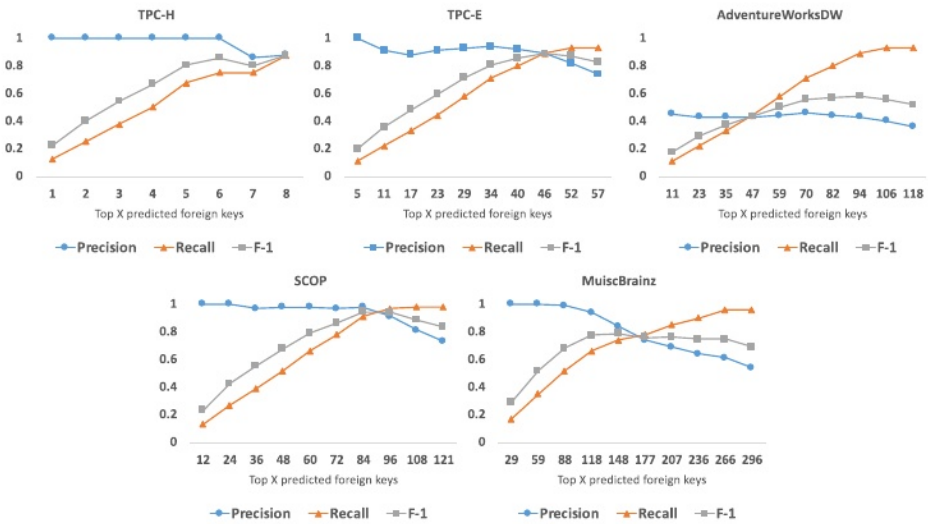


图 5 HoPF 在五个数据集上的有效性。x 轴表示选择前 X 个预测的外键

表 4 采用和未采用主键缩减时预测出的真实主键和真实外键

数据集	# 有（无）降解的 PKs w	# 带（不带）约简的 FKs w
TPC-H	8 (8)	7 (7)
TPC-E（事务处理性能评估基准）	27 (30)	42 (40)
AdvWorks	25 (25)	40 (42)
SCOP	35 (41)	88 (77)
MusicBrainz	93 (109)	161 (137)

错误的主键 错误预测的主键可能会极大地损害外键发现的结果，因为外键必须引用特定的主键。因此，一旦 HoPF 获得了错误的正负主键，它可能会预测出多个错误的正负外键。但在实践中，我们并未目睹由错误主键导致的很多错误。在检查结果后，我们发现那些虚假的主键候选通常得分较低，因而会被冲突规则和连通性限制过滤掉。

空的左值列 如果包含在 IND 中的左值为空，则左值中的所有属性仅包含空值。具有空左值的 IND 仍可以是有效的外键，尤其是在现实世界的数据集中。然而，我们的数据分布特征无法对这种包含依赖关系进行评估，因此我们在早期阶段将其过滤掉。例如，在 *MusicBrainz* 中，artist *type.parent* ⊆ artist type.*id* 是一个真实的外键，但由于左值为空列，所以无法检测到。我们在 *MusicBrainz 中仅观察到 24 个具有空左值的其他外键*，这大大降低了召回率。

尽管我们在早期阶段使用 *PK* ⊆PK 过滤器来移除这些候选键，但在算法通过主键缩减移除冗余主键时，其中一些可能会被重新添加到预测结果中。例如，在 *MusicBrainz* 中，*place alias type.id* ⊆label *alias type.id* 在算法不再将 {place *alias type.id}* 视为主键后，会被重新考虑为外键。然而，从表 4 中可以看出，由于主键缩减在大多数情况下效果良好，HoPF 仍然能够阻止大多数 *PK* ⊆*PK* 候选键。

通过移除那些被一些得分较低的预测外键所引用的主键，所发现的外键会获得更高的总体得分。因此，HoPF 将其视为非主键。然而，主键发现性能的损失提高了外键发现的性能，如图所示

表 5 金标准以及 HoPF 发现的主键和外键数量，分别标记为“true”和“disc.”。“undoc.”代表 HoPF 发现的未记录的外键。

数据集	真实对决	讨论：对决	真实外键	附注。FKs	未记录的。FKs
TPC-H	8	8	8	7	0
TPC-E（事务处理性能评估基准）	27	27	45	42	0
AdvWorks	27	25	45	40	2
SCOP	42	35	90	88	2
MusicBrainz	124	101	168	161	0

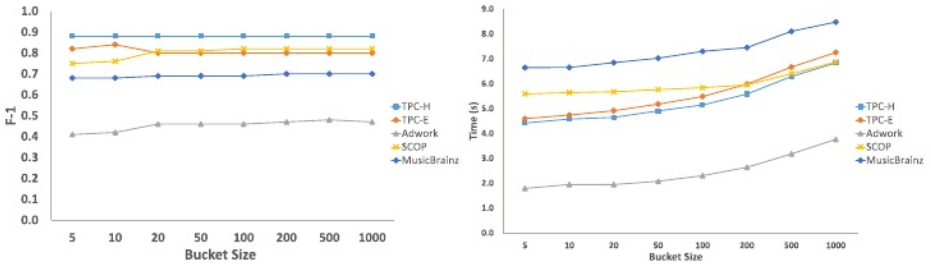
如下所述。它恢复了在上一步中被视为 $PK \subseteq PK$ 的一组真正的外键（表 5）。

表 4 展示了 HoPF 在应用和未应用主键缩减的情况下预测出的真正主键和外键的数量。例如，对于 *SCOP* 数据集，应用主键缩减后 HoPF 预测出 35 个主键和 88 个外键；未应用主键缩减时，相应的数量分别为 41 个主键和 77 个外键。从结果可以看出，除了 AdvWorks 数据集外，应用主键缩减策略有助于预测出更多的外键，但代价是会丢失一些真正的主键。对于 *AdvWorks* 数据集，在应用主键缩减后，预测中少了两个真正的外键。仔细检查结果后我们发现，主键缩减策略错误地移除了一个具有单列主键的表的真正主键，而该主键被两个丢失的外键所引用。因此，我们建议对于包含未定义主键的连接表的数据集，采用主键缩减策略，这样可以在少量丢失真正主键的情况下发现更多的外键。然而，我们认为用户更容易注意到缺失的主键，而不是找出缺失的外键，因为后者需要综合考虑多个表。在我们的实验中，我们对所有数据集都采用了主键缩减的方法。

我们还测试了选择不同桶数对直方图差异特征的影响。图 6a 展示了每个数据集在不同桶数下的 F-1 分数。当桶数小于 20 时，分数会有小幅波动，但总体保持稳定。图 6b 显示了每个数据集的时间消耗。为了便于阅读，时间以秒为单位并以对数形式显示。可以看出，使用更多桶来构建此数据结构的时间开销会显著增加。对于所有参与的数据集，最优桶数在 10 到 20 之间，我们选择 20 作为默认值。

仅使用最小的唯一候选键（UCC）作为 HoPF 的输入来生成主键候选，可能会遗漏真正的 n 元主键，如果 n 元主键的一个子集已经是 UCC 的话。这一问题及其对连接表造成的影响在本节前面已有所讨论，为此我们提出了主键缩减策略。此外，对于非常小的数据集，即每个关系中只有少量记录的情况，缺乏完整的 UCC 集合也会导致类似的结果，因为在这种情况下，关系更有可能具有单元 UCC，从而掩盖了作为真正主键的更大 UCC，因为我们仅将最小的 UCC 作为输入。

为了衡量这种影响，我们在规模因子为 0.001 的 TPC-H 实例上应用 HoPF。在这种情况下，数据集中的所有关系仅包含几十条记录。结果证实了我们的怀疑：HoPF 无法发现表 *lineitem* 和 *partsupp* 的真实主键。



(a) F-1 score under different bucket number.

(b) Time consumption under different bucket number.

图 6 不同桶数下的 F-1 分数及其对应所花费的时间

具有 n 元主键，因为它们都具有作为真实主键一部分的一元唯一列组合。不幸的是，我们的算法在不改变仅使用最小唯一列组合这一前提条件的情况下无法解决这种小表困境。正如第 3.2 节所述，改变前提条件以使用非最小唯一列组合也无法解决这一困境，因为所提出的主键特征倾向于选择具有较小基数的唯一列组合候选，因此 HoPF 更有可能将它们视为真实主键。

7.3 无主键的外键检测

以往的研究都假定主键已存在。然而，对于许多数据库而言，尤其是那些以转储文件或平面文件形式存储的数据库，这种假设过于乐观，因为这些数据库的约束定义可能并未与数据本身紧密关联。在这种情况下，基于该假设的外键发现算法可能会失效，或者预测出大量错误的外键。在这里，我们使用 HoPF 来探究在没有主键的情况下对发现的外键质量的影响。幸运的是，即使没有发现主键，我们的方法仍能根据外键得分对 IND 列表进行排序。HoPF 不是通过外键整体预测主键，而是只要其右值集是唯一候选码（UCC），就输出预测的外键。

表 6 展示了在已知和未知主键的情况下检测到的外键。与我们的假设一致，在不知道主键的情况下，HoPF 获得的正确外键数量更少，但代价是在更大的外键候选集中进行搜索。例如，在 *Musicbrainz* 中，如果我们忽略主键，仅根据分数对候选外键进行排序，从 1079 个候选外键中获得 141 个正确外键；而如果我们先检测主键并将其作为外键检测的输入，则从仅 296 个候选外键中获得 161 个正确外键。

通常，我们仅将右部属于主键集的包含依赖关系纳入外键候选集。若不了解主键，HoPF 会将每个右部为唯一候选键的包含依赖关系视为外键候选。这解释了为何在不考虑主键的情况下外键候选的数量会增加；每个表通常包含许多唯一候选键，而每个唯一候选键都可能为候选集贡献多个有效的包含依赖关系。

由于*外键*规则的*唯一性*，如果另一个外键候选的得分更高且被预测为外键，则与之具有相同左部的真正外键将被排除在外。然而，如果预测的外键不正确，将真正的外键添加到最终结果中可能会导致循环引用，从而被拒绝。这解释了为何在不了解主键的情况下获取的真正外键较少。

表 6 未考虑和考虑主键时检测到的外键

数据集	FKs	无主键的外键索引	候选键无主键	FK 与 PK 的差异	带主键的候选键
TPC-H	8	2	18	7	8
TPC-E（事务处理性能评估基准）	46	45	77	42	57
AdvWorks	45	39	369	40	118
SCOP	90	88	167	88	121
MusicBrainz	168	141	1,079	161	296

7.4 未记录的外键发现

除了发现已记录的外键之外，在 *AdventureWorksDW* 和 *SCOP* 中还发现了其他潜在的外键，尽管这些外键在黄金标准中并不存在。这些不确定的外键分为两类：缺失的外键和错误的外键。例如，在 *SCOP* 中，*pdb release author.pdb author id* ⊆ pdb author.id 被视为误报，但我们认为它实际上是一个真正的外键。另一方面，cdd.*release id* ⊆ *pfam* release.*id* 是一个已记录的外键，但我们认为其记录有误；正确的外键应该是 cdd.*release id* ⊆ cdd release.*id*，这是由 HoPF 预测得出的。

我们可以设想出这些未记录的外键存在的几种原因，例如在数据迁移过程中丢失，或者被模式设计者为了提高查询效率而移除。表 5 的最后一列显示了 HoPF 预测的每个数据集中的未记录外键数量。这样的发现可能会为我们进一步的数据库应用（如数据集成）提供一些启示。

7.5 比较

我们重新实现了 Zhang 等人（2010 年）和 Chen 等人（2014 年）的最新算法，分别将其称为“*随机性*算法”和“*FastFK*”，并将其性能与 HoPF 进行了比较。正如第 2 节中更详细解释的那样，*随机性*算法

表 7 HoPF 与其他两项先前工作（Zhang 等人 2010 年；Chen 等人 2014 年）在检测外键方面的比较。加粗数字表示该算法相对于其他算法的更优性能。

数据集	真实外键	算法	带列名				无列名			
			P	R	F-1	预测的外键	P	R	F-1	预测的外键
TPC-H	8	快速正向运动学	.56	.90	.69	16	.56	.90	.69	16
		随机性	**1**	**1**	**1**	8	.21	**1**	.35	39
		霍普夫	.88	.88	.88	8	**.88**	.88	**.88**	8
TPC-E（事务处理性能评估基准）	104	快速正向运动学	.72	**.95**	.82	59	.59	.78	.67	59
		随机性	**1**	.89	**.94**	45	.57	**.82**	.67	308
		霍普夫	.72	.91	.80	57	**.64**	**.82**	**.72**	57
AdvWorks	45	FastFK	.32	**.97**	.49	131	.24	**.72**	.37	131
		随机性	**.90**	.41	**.56**	122	.21	.58	.31	122
		霍普夫	.31	.84	.46	118	**.27**	**.72**	**.39**	120
SCOP	90	FastFK	.57	**.94**	.71	149	.53	**.87**	.66	149
		随机性	.36	.61	.45	151	.36	.61	.45	151
		霍普夫	**.70**	**.94**	**.80**	121	**.63**	.82	**.71**	118
MusicBrainz	168	FastFK	.33	.74	.46	368	**.28**	**.62**	**.39**	367
		随机性	.24	.49	.32	341	.24	.49	.32	341
		霍普夫	**.54**	**.96**	**.69**	296	**.28**	.50	.36	289

在 HoPF 和 Chen 等人（2014 年）的研究中，将外键候选的左外键（LHS）和右外键（RHS）之间的列名相似度设为外键特征，而非匹配的列名则用于后期处理（Zhang 等人，2010 年）。我们使用四个指标来衡量性能：准确率（P）、召回率（R）、F1 值以及预测的外键数量（预测的外键）。

FastFK 假定仅存在单属性外键，它采用一些特征来对候选外键进行评分，并使用一些剪枝规则来减少搜索空间。对于*随机性*度量，我们采用 θ = 0.9，对于一元和 n 元外键候选分别使用 256 个底图、256 个和 16 个分位数——这是原作者确定的最佳值。由于这两项先前的工作都假定主键存在且已知，因此在本实验设置中我们为它们提供了真实的主键。

为了提高结果的准确性，Randomness 还会考虑列名，仅保留列名完全匹配的候选对象。作者仅将此技术应用于 TPC-H 和 TPC-E，依靠外部文档手动修剪列标签，然后再进行匹配。我们比较了在启用*列名*特征的情况下 Randomness 与 HoPF 和 *FastFK* 的结果。我们还比较了未进行此后期处理的 Randomness 在禁用*列名*特征时的结果与另外两种方法的结果。表 7 展示了这些比较的详细情况。

如表所示，在不了解列名的情况下，这三种算法的准确率和召回率均有所下降，这证明列名对于识别外键具有指示作用。在使用列名的情况下，这三种算法在合成数据集（TPC-H、*TPC-E* 和 *AdventureWorks*）上的表现相当，而在真实世界的数据集（SCOP 和 *MusicBrainz*）中，*HoPF 的表现优于两种基线方法*。我们还注意到，与两种基线方法相比，HoPF 对所有数据集生成的预测外键集都更小，这使得任何后续的人工专家处理都更加容易。

8 结论

主键和外键是保持数据库一致性的关键完整性约束。然而，以平面文件或转储形式存储的数据并不总是包含这些约束定义，在很多情况下，识别这些约束的任务就落在了数据使用者身上，从而更好地理解数据并确保其质量。由于模式可能相当庞大且复杂，自动发现主键和外键是一个相关（且具有挑战性）的研究课题。

此前的研究分别尝试发现外键和主键。在这项工作中，我们提出了 HoPF 算法，以整体方式整合主键和外键的检测。我们采用一组精心设计的特征来对真正的主键和外键与虚假的唯一组合键（UCC）和独立列（IND）进行评分和区分。我们还采用了若干有用的剪枝规则，以有效减少主键和外键的搜索空间。

在对五个不同数据集进行的性能实验中，我们的算法在主键和外键发现方面分别达到了平均召回率 88% 和 91%。通过一项实验我们表明，在没有主键知识（这是相关工作的假设）的情况下，外键发现的性能要差得多，这表明有必要提前或同时发现主键。我们还与最先进的算法进行了精度和召回率的比较。

作为未来的工作，我们计划将关于选择良好主键和外键的工作与第 2.1 节中提到的基于格的 UCC 和 IND 检测算法相结合，为它们庞大的搜索空间提供一种强大的（但不精确的）剪枝机制。

参考文献

Abedjan, Z., Golab, L., Naumann, F. (2015). Profiling relational data: a survey. *VLDB Journal*, *24*(4), 557–581.

Bhattacharyya, A. (1943). On a measure of divergence between two statistical populations defined by their probability distributions. *Bulletin of the Calcutta Mathematical Society*, *35*, 99–109.

Chaudhuri, S., Ganjam, K., Ganti, V., Motwani, R. (2003). Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the international conference on management of data (SIGMOD)* (pp. 313–324).

Chen, Z., Narasayya, V.R., Chaudhuri, S. (2014). Fast foreign-key detection in microsoft SQL server powerpivot for excel. *Proceedings of the VLDB Endowment*, *7*(13), 1417–1428.

Faust, M., Schwalb, D., Plattner, H. (2014). Composite group-keys –space-efficient indexing of multiple columns for compressed in-memory column stores. In *Memory data management and analysis - first and second international workshops, revised selected papers* (pp. 139–150).

Ilyas, I.F., Markl, V., Haas, P.J., Brown, P., Aboulnaga, A. (2004). CORDS: automatic discovery of correlations and soft functional dependencies. In *Proceedings of the international conference on management of data (SIGMOD)* (pp. 647–658).

Kantola, M., Mannila, H., R¨aih¨a, K., Siirtola, H. (1992). Discovering functional and inclusion dependencies in relational databases. *International Journal of Intelligence Systems*, *7*(7), 591–607.

Lopes, S., Petit, J., Toumani, F. (2002). Discovering interesting inclusion dependencies: application to logical database tuning. *Information Systems (IS)*, *27*(1), 1–19.

Lucchesi, C.L., & Osborn, S.L. (1978). Candidate keys for relations. *Journal of Computer and System Sciences*, *17*(2), 270–279.

Marchi, F.D., Lopes, S., Petit, J. (2009). Unary and n-ary inclusion dependency discovery in relational databases. *Journal of Intelligent Information System*, *32*(1), 53–73.

Memari, M., Link, S., Dobbie, G. (2015). SQL data profiling of foreign keys. In *Proceedings of the international conference on conceptual modeling (ER)* (pp. 229–243).

Papenbrock, T., Bergmann, T., Finke, M., Zwiener, J., Naumann, F. (2015). Data profiling with metanome. *Proceedings of the VLDB Endowment*, *8*(12), 1860–1863.

Papenbrock, T., & Naumann, F. (2017). Data-driven schema normalization. In *Proceedings of the international conference on extending database technology (EDBT)* (pp. 342–353).

Rostin, A., Albrecht, O., Bauckmann, J., Naumann, F., Leser, U. (2009). A machine learning approach to foreign key discovery. In *Proceedings of the ACM SIGMOD workshop on the web and databases (WebDB)*.

Rubner, Y., Tomasi, C., Guibas, L.J. (1998). A metric for distributions with applications to image databases. In *Procedings of the international conference on computer vision (ICCV)* (pp. 59–66).

Tschirschnitz, F., Papenbrock, T., Naumann, F. (2017). Detecting inclusion dependencies on very many tables. *ACM Transactions on Database Systems (TODS)*, *42*(3), 18:1–18,29.

Venetis, P., Halevy, A.Y., Madhavan, J., Pasca, M., Shen, W., Wu, F., Miao, G., Wu, C. (2011). Recovering semantics of tables on the web. *Proceedings of the VLDB Endowment*, *4*(9), 528–538.

Zhang, M., Hadjieleftheriou, M., Ooi, B.C., Procopiuc, C.M., Srivastava, D. (2010). On multi-column foreign key discovery. *Proceedings of the VLDB Endowment*, *3*(1–2), 805–814.

出版者注：施普林格·自然对于所发表的地图中涉及的管辖权主张以及机构隶属关系保持中立。