Desbordante：从基准测试套件到高性能科学密集型数据剖析器（预印本）

George Chernishev[1,2], Michael Polyntsov[2], AntonChizhov[2], Kirill Stupakov[2], IlyaShchuckin[2], Alexander Smirnov[2],Maxim Strutovsky[1,2], Alexey Shlyonskikh[2], Mikhail Firsov[2], Stepan Manannikov[2],Nikita Bobrov[2],Daniil Goncharov[2], Ilia Barutkin[2], Vladislav Shalnev[2], Kirill Muraviev[2], AnnaRakhmukova[2], Dmitriy Shcheka[2], AntonChernikov[2], DmitriiMandelshtam[2],MikhailVyrodov[2], ArthurSaliou[2], Eduard Gaisin[2], Kirill Smirnov[2]

[1]Unidata LLC ，俄罗斯

[2]圣彼得堡大学，俄罗斯

圣彼得堡，俄罗斯

georgii.chernyshev@unidata-platform.ru,polyntsov.m@gmail.com,anton.i.chizhov@gmail.com,kirill.v.stupakov@gmail.com,shchuckinilya@gmail.com,alexander.a.smirnovv@gmail.com,maxim.strutovskiy@unidata-platform.ru,shlyonskikh.alexey@gmail.com,mikhail.a.firsov@gmail.com,manannikov.st@gmail.com,nikita.v.bobrov@gmail.com,goncharov.y.daniil@gmail.com,ilia.d.barutkin@gmail.com,vladislav.a.shalnevv@gmail.com,rakhmukova.anna@gmail.com,dmitriy.v.shcheka@gmail.com,chernikov.a.anton@gmail.com,mandelshtamd@yandex.ru,mikhail.v.vyrodov@gmail.com,arthur.salio@gmail.com,edu.gaisin@gmail.com,kirill.k.smirnov@gmail.com

arXiv:2301.05965v1 [cs.DB] 14 Jan 2023

摘要

Metanome 和 Open-Clean 等开创性的数据剖析系统，使科学密集型数据剖析受到公众关注。这类剖析旨在提取复杂模式（原语），例如函数依赖、数据约束、关联规则等。然而，这些工具是研究原型，而非可投入生产的系统。

本文介绍了 Desbordante——一款高性能、科学密集型数据分析器，采用开源代码。与类似系统不同，它的设计侧重于在多用户环境中的工业应用。它高效、崩溃后恢复能力强且可扩展。其效率得益于使用 C++ 实现发现算法；其韧性通过广泛采用容器化实现；其可扩展性则基于容器复制。

Desbordante 旨在向更广泛的公众开放工业级基元发现，重点面向并非 IT 专业人员的领域专家。除了发现各种基元，Desbordante 还提供基元验证：它不仅会报告给定基元实例是否成立，还会通过专用屏幕指出阻碍其成立的因素。接下来，Desbordante 支持流水线——使用已发现基元实现的即用型功能，例如拼写错误检测。我们提供内置流水线，用户也可以通过提供的 Python 绑定构建自己的流水线。与其他数据分析器不同，Desbordante 不仅适用于表格数据，也适用于图数据和事务数据。

本文介绍了 Desbordante、其背后的愿景及其应用场景。为提供更深入的视角，我们将讨论它的当前状态、架构以及构建时所依据的设计决策。此外，我们还将概述未来计划。

1 引言

根据 [2]，，数据剖析是“用于确定给定数据集元数据的一系列活动和流程”。此类元数据可用于数据集探索、与数据质量、数据库管理和数据库逆向工程相关的各种任务。它在数据集成和查询优化领域也有许多应用 [3, 20] 。

数据剖析可分为朴素剖析和科学密集型剖析。朴素剖析关注提取简单事实，例如表中的行数和列数、某列中的最小值和最大值、检测原子列数据类型等。属于这一类的数据剖析工具有数百种，因为几乎所有大型信息系统供应商都提供此类工具。也有许多开源工具，其中最知名的之一是 Pandas Profiling [11]。

另一方面，科学密集型分析器侧重于提取复杂元数据，这需要复杂的算法。此类元数据发现包括：从表格中提取各种函数依赖（既包括精确的，也包括宽松的） [12], 关联规则挖掘 [4],语义列类型检测 [19], 数据约束发现 [6, 37]等等。提供此类数据剖析功能的工具要少得多。提供此类功能的两个重要系统是 Metanome [26] 和 OpenClean [24]。

本文介绍 Desbordante（西班牙语意为无边界）——一款高性能科学密集型数据分析器。它的灵感源自 Metanome，但在多个关键方面与之不同。首先，Desbordante 专注于多用户环境下的工业应用。提取复杂元数据需要复杂的算法，而这些算法的计算成本极高，也容易发生崩溃。Desbordante 通过审慎地实现算法，并采用专门设计的应用架构来应对这些问题。

其次，我们对这类工具的使用场景有着不同的设想。我们设想，主要用户是拥有大量数据并希望对其进行探索的领域专家，同时他们不一定是 IT 专业人员。用户希望发现数据中各种能够揭示非平凡事实的模式。这些模式通过多种所谓的原语进行形式化描述。原语本身是对一条适用于数据（或其中一部分）的规则的描述，并以数学方法进行形式化表述。函数依赖就是一个众所周知的例子。

可能对模式发现感兴趣的专家类型包括：

（1）生物信息学研究人员、化学家、地质学家，事实上，几乎所有处理大量数据的科学家，尤其是处理实验所得数据的科学家。（2）从事金融数据相关工作的人：金融分析师、销售人员和交易员，他们手头也都有大量可供探索的数据。（3）数据科学家、数据分析师和机器学习专家。

对于处理大量数据的科学家来说，发现一个基元意味着某种模式的存在。基于此，他们或许能够提出假设，甚至立即得出结论（如果数据足够多）。至少，发现的模式可以为他们的后续研究指明方向。例如，圣彼得堡 JetBrains 实验室的生物信息学小组在其工作中使用了这类基元 [35]。

对于金融数据，研究人员也可以尝试提出某种假设（例如，“在竞争公司提供的所有汽车中，红色汽车最畅销”）。不过，还有一些更常见、需求更大的应用场景：清理数据中的错误、查找并删除近似重复项，等等。请注意，科学家可能也会对这项功能性感兴趣，不过兴趣要小得多。

对于机器学习，发现的基元有助于特征工程，以及选择消融研究的方向。

上述用例对我们的基元发现工具提出了以下具体要求：

- 聚焦于近似原语。我们的用户处理的是实际数据，因此实现近似原语应当优先考虑。
- 不仅要聚焦于基元发现，还要关注基元验证和结果的可解释性。我们需要向用户说明，为什么某个基元实例不成立。• 聚焦于可调节性。用户希望通过指定各种约束，对发现过程进行微调，以找到所需的基元。• 聚焦于非表格数据。尽管表格是最常见的数据类型，用户也对图和事务数据等其他类型感兴趣。• 聚焦于支持多种接口。部分用户需要控制台应用程序，部分需要功能丰富的网页用户界面，还有部分需要 Python 接口。

学术数据库领域已经提出了大量基元，用于描述数据中可能存在的各种模式。例如，仅针对宽松函数依赖这一类，就有三十多种不同的定义方式 [12]。而且每年都有新的基元不断出现。然而，领域之外的人大多并不了解这些基元。最糟糕的情况下，它们只会停留在理论成果阶段；即便情况较好，也只是以鲜为人知的原型形式存在，尚不能用于工业应用。还有一些工具 [24, 26] 包含用于发现和验证基元的算法集合，但它们同样不具备可用于生产环境的条件。它们受性能限制，而且缺乏所需功能，无法满足用户的特定需求。

因此，Desbordante 的理念是向公众“开放”这些基元，让每个人都有机会研究自己的数据。

现在，转而介绍 Desbordante 本身：

- 工业应用导向体现在效率和韧性上。与其他科学密集型分析器不同，Desbordante 的内核是用 C++ 实现的。
- Desbordante 提供控制台和网页版。此外，还提供 Python 接口。
- 它支持流水线——利用发现的基元实现的即用型功能，例如错别字检测。用户可以通过 Python 绑定构建自己的流水线。
- 它支持表格数据和非表格数据（图数据、事务数据）。

Desbordante 是一个[1] 采用现代技术栈实现的开源项目。你可以在此处[2]体验已部署的演示版。

本文主要介绍工具愿景、架构，以及我们为满足用户需求并确保高性能、韧性和可扩展性所采用的方法。我们还讨论了工具定位，列出了目前支持的基元，并介绍了未来计划。

本文结构如下。第 2 节介绍相关工作，并讨论现有的数据剖析工具。接下来，第 3 节将描述基元及其发现所特有的要求。第 4 节将从宏观和微观角度审视 Desbordante，介绍项目愿景、核心功能、有关用户体验和系统性能的考量。第 5 节将概述 Desbordante 的系统架构、技术栈，以及已实现微服务的实现方式和原因。第 6 节将列出我们的未来计划及其里程碑。

2 相关工作

现有工具很难按照属性、功能性或愿景进行分类，使其与 Desbordante 完全对应。因此，我们回顾了最知名的工具；这些工具实现了多种数据剖析技术，能够挖掘基元，或至少高度依赖基元。

在数据分析器领域，与 Desbordante 最接近的是 Metanome [26]。简而言之，Metanome 是一个框架，为开发者提供基础设施和一套相应的接口，用于实现基元并对其进行基准测试。Metanome 的架构使研究构想的开发和测试过程尽可能快捷、便利，从而让开发者能够专注于算法，而不必编写数据库连接器、文本处理工具等样板代码。Metanome 由哈索·普拉特纳研究所的团队开发，该团队开发的几乎所有算法都可以作为 JAR 文件接入 Metanome（将算法视为可连接的编译代码这一概念，是 Desbordante 目前所欠缺的）。然而，由于我们在对两个平台进行广泛评估和比较时所述的原因，Metanome 不能被视为 Desbordante 真正的工业级替代方案 [34]：例如，它的内存占用更大，依赖发现速度更慢。尽管如此，Metanome 系统仍启发了 Desbordante，使其成为一款用户友好、高性能且灵活的数据分析器。

OpenClean [24] 系统的目标是通过专注于数据清洗和数据剖析这一细分领域，成为现代数据科学技术栈的一部分。作为一个开源 Python 库，OpenClean 为用户提供了一个环境，使他们能够将数据剖析无缝集成到其他框架和库中，用于

[1] https://github.com/Mstrutov/Desbordante（算法），https://github.com/vs9h/Desbordante（Web 应用程序）。
[2] https://desbordante.unidata-platform.ru

数据处理和机器学习。函数依赖（FD）的概念有两种用途：检查数据是否违反函数依赖，以及进行函数依赖挖掘。前者在 Python 中通过映射器与分组筛选器的组合实现（类似于在 SQL 中用于检查函数依赖的 **SELECT. . . GROUP BY . . . HAVING** 惯用写法）。对于违反函数依赖的记录，可以通过 OpenClean 的修复策略或用户自定义策略启动修复流程。基元挖掘功能由一个独立软件包提供，该软件包会启动一个子进程来运行 Metanome JAR 文件。基本上，所有曾为 Metanome 实现的算法都可以在 OpenClean 中运行。这也意味着，就算法性能而言，OpenClean 继承了 Metanome 的所有问题。

与 OpenClean 不同，大多数数据清洗工具都没有内置的基元挖掘功能，并且要求用户将基元作为输入数据提供给清洗流程 [7, 13, 29, 30]。例如，数据修复框架 HoloClean [29] 充分利用了否定约束 (DC)，而否定约束涵盖了函数依赖 (FD)、条件函数依赖 (CFD) 和匹配函数依赖等概念。HoloClean 的输入包括一个不一致数据集、一组否定约束，以及可用于修复该数据集的任何外部知识。HoloClean 会综合利用所有可用信息，并提出能够使数据恢复一致状态的解决方案。由于 HoloClean 的主要目标是为谨慎地恢复数据集的一致形式铺平道路，因此该工具没有实现任何用于基元挖掘或高效原位推断元数据的内部机制。Horizon [30], 也秉持相同理念：它根据给定的函数依赖集计算函数依赖模式图，并通过模式图静态分析构建解决方案。

然而，并非所有数据剖析工具都认为基元对数据清理流程或错误检测任务非常重要。否则，它们会依赖机器学习、概率方法 [36], 或精选知识库，就像 KATARA 所做的那样 [14]。KATARA 的作者甚至拒绝将函数依赖视为可信元数据，因为这类原语无法保证以明确无歧义的方式修复数据。

一些数据工程工具遵循不同的理念：它们不修复数据，而是确保数据一开始就整洁。Great Expectations [1] 允许用户定义复杂的完整性约束，并在数据流经 ETL 流程时将其用作断言机制。这类用于数据验证的微型单元测试可以嵌入工作流中；一旦发生任何意外情况，例如新到达的数据违反了 Great Expectations 框架中描述的某个基元，系统便会立即发出警报。Auto-Validate [32] 系统也采用了类似的理念。

上述工具是研究成果的实现形式，这些成果已在数十篇论文中得到仔细梳理，其中一些工具是开源的，可免费使用。为使概述更加完整，我们还想列举一些工业解决方案。

大多数商业解决方案都将函数依赖（FD）作为数据剖析的一部分提供支持：SAP Information Steward、Oracle Warehouse Builder、Informatica Data Quality、Microsoft SQL Server Data Profiling Task 和 Talend Open Studio 都可以返回在数据上近似成立的函数依赖，或验证用户指定的依赖是否成立。对于每个近似函数依赖（AFD），这些工具还会记录违反该依赖的记录比例。这种处理函数依赖/近似函数依赖的方式似乎已成为任何数据剖析工具几乎必备的功能，在执行异常检测和探索“损坏”的记录时也很有用。然而，通常函数依赖/近似函数依赖是唯一被付费工具实现的原语，因为如今这类工具的关注重点已转向数据剖析领域中的机器学习方向。

3 科学密集型基元的发现

3.1 现状与动机

单就其本身而言，原语是对某种规则（模式）的描述，该规则以数学方式描述并适用于数据。函数依赖就是一个很好的例子：依赖关系 A $\longrightarrow$ B（A 和 B 是列）成立，当且仅当对于每一对行，A 中的值相等都意味着 B 中的值相等。

原语有数百种类型 [12, 33],，每一种都有成熟的属性和坚实的理论基础。此外，新类型也一直在开发中。

然而，正如引言中所述，它们大多局限于数据库及相关领域，无法让更广泛的公众受益。

原因如下：

- 总体而言，基元的实现难以获取，甚至根本无法获取：– 大多数基元都是在 GitHub 尚未普及的年代开发的，当时作者通常不会提供源代码，或者源代码会发布在研究小组的网站上，而这些网站如今往往早已不复存在。无论是哪种情况，目前都无法获取源代码。– 如今能够获取的实现散落在网络各处，可能位于个人网站或鲜为人知的代码库中。当然，介绍该基元的论文通常会附上链接，但潜在用户必须先知道该基元及其论文，而事实并非如此。

- 即使能够获取，也很难完成配置和运行 .
  例如，我们团队的新成员花了 6 到 12 个工作小时才完成 Metanome 的配置和运行。同时，他们大多是熟悉 IT 专业知识的计算机科学专业学生。因此，对于想尝试某种基元的非 IT 专业人士来说，这会是一项相当繁琐的任务。
- 每一种可用的原语实现（甚至是发现算法实现）都需要各自的软件生态系统来进行搭建和维护。这对于想要尝试某些原语的非技术用户来说，又是一道需要克服的障碍。
- 最后，可用的实现通常只是为某篇论文编写的概念验证或原型，之后便遭到弃用。因此，它们通常效率不高，因为开发时采用了便于快速构建原型的语言，例如 Java 或 Python。这些语言缺乏 C++ 的效率和底层可调性。此外，还存在可扩展性问题，因为真实世界数据集很可能比论文作者用于基准测试的数据集更大。而且，这些实现处理作者未用于基准测试的数据集时可能会崩溃。因此，有必要进一步拓展适用范围的边界（这也正是 Desbordante 这个名称的由来）。

有一些平台尝试解决这些问题，例如 Metanome 或 OpenClean。然而，它们无法同时解决所有这些问题。因此，我们需要一个工业级的

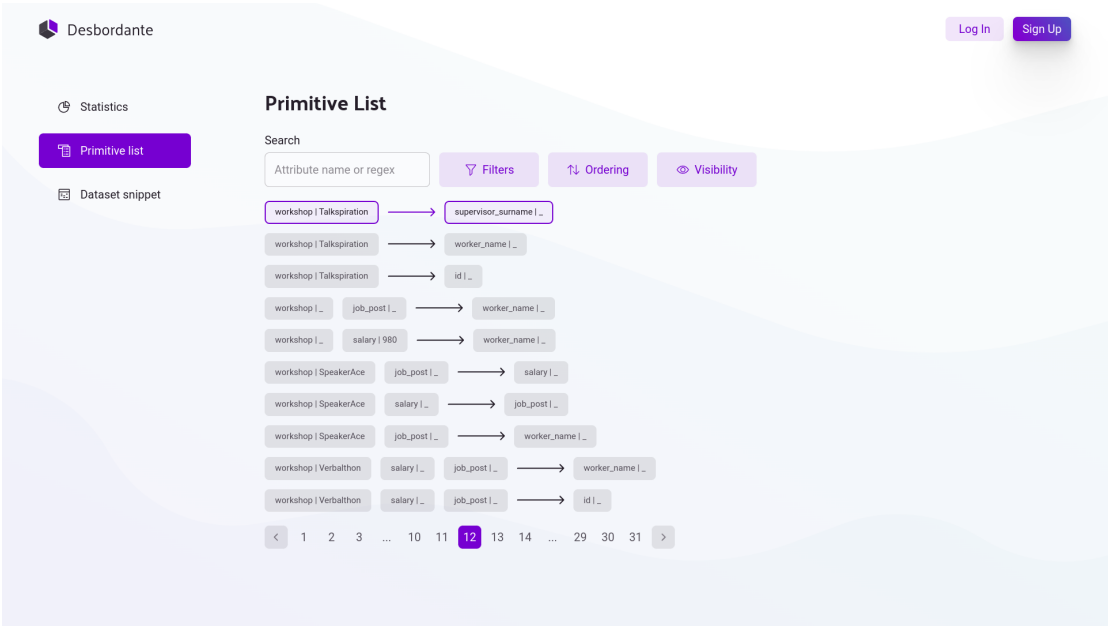

图 1：Desbordante 网页版的结果页面（条件函数依赖）

平台，让更广泛的公众能够进行原语发现，而 Desbordante 正是为实现这一目标而努力。

3.2 科学密集型原语发现的具体特点

科学密集型基元的发现有其自身的特点，可从几个方面进行描述，并分为两组。第一组的特点是所有科学密集型分析器共有的，源于原语发现任务本身的性质。

（1）原语发现是一项计算难度很高的问题 .
发现算法即使面对小型数据集，也会遇到时间或内存限制。例如，参见 [28] 中的表 1，其中列出了使用服务器级硬件挖掘函数依赖时可处理的数据集大小。除两个数据集外，其余数据集均小于 1 MB。其他基元的情况也类似。因此，要让基元发现真正具备实用性，我们需要解决这些限制。

（2）发现算法的实现非常复杂，通常依赖第三方库，而且通常解决的是科学前沿领域中的问题。因此，这些实现很脆弱——面对某些输入时可能会崩溃或卡死。因此，在将其“工业化”时，必须通过提高应用程序的容错能力来提升其可靠性。

第二组描述的是与我们系统的愿景和目标相关的方面，体现了用户的使用场景和需求。

(1) 用户更关注近似原语。真实世界数据很可能包含各种错误、缺失值以及其他类型的伪影。因此，精确版本的原语并不适用，在真实数据中也很少见。相反，在开发我们的分析器时，我们必须提供非精确版本，以允许一定程度的误差。

(2) 用户不仅需要发现原语，还需要验证原语。与发现不同，验证会将某个特定的原语实例（例如，特定的函数依赖）作为输入，并返回该原语是否成立。这意味着需要提供专用屏幕，让用户能够分析数据，并查看哪些因素导致给定的原语不成立（例如，冲突值、行等）。

(3) 我们需要为用户提供各种调节参数，以控制发现过程。例如，在发现函数依赖时，众所周知，左侧较大的依赖价值较低。发现这类依赖通常并不意味着存在真正的依赖，而是表明用于挖掘的数据片段太小，无法包含反例。原语发现过程总是代价高昂，因此跳过不必要的计算是值得的。另一个重要例子是为近似原语设置误差阈值。与此同时，合适的取值取决于特定的数据集和用户目标。

(4) 用户对所使用的接口有不同偏好。有些人喜欢传统的命令行接口，而另一些人则希望使用功能丰富的网页用户界面。此外，要让数据科学家能够使用原语，提供 Python 接口至关重要。

Desbordante 旨在将这些具体情况纳入考量。

4 DESBORDANTE

4.1 概述

Desbordante 的核心是一个 C++ 库，其中包含原语发现算法所需的所有辅助数据结构、算法本身以及所有必需的配套基础设施。该库提供了用于执行算法并获取其结果的 API，供 Web 应用程序的后端使用。此外，还有一个提供 Python 绑定的附加库版本，因此可以在 Python 中使用 Desbordante 的所有功能

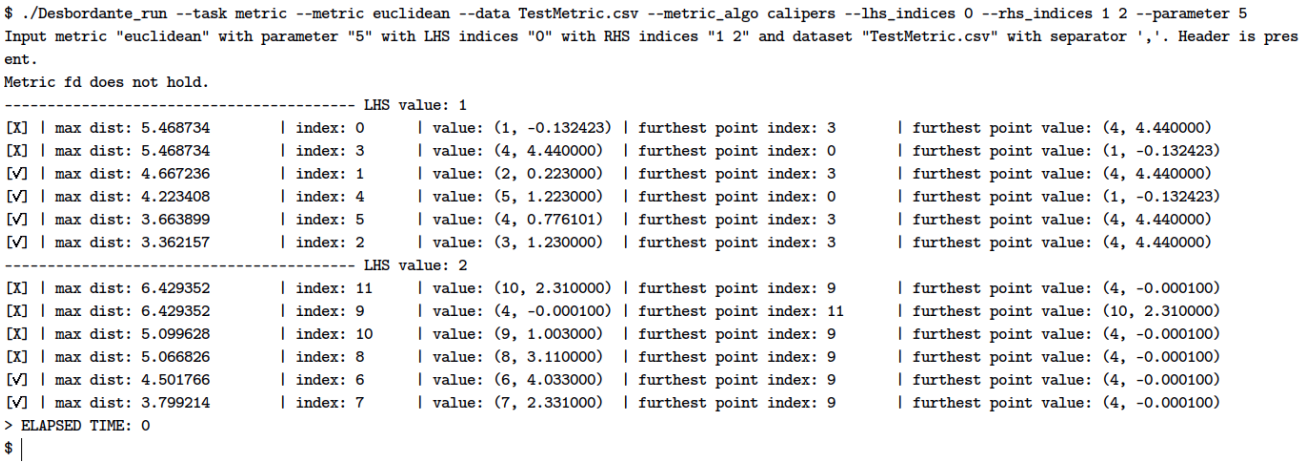

图2：度量函数依赖的控制台验证界面

。Desbordante 库提供两种用户界面：控制台和 Web 界面。

Desbordante 支持两种不同类型的任务（以及相应的算法）：

- 发现：在指定数据集上查找指定原语的所有成立实例；
- 验证：给定一个原语实例，判断它是否在指定数据集上成立；若不成立，则提供有关其不成立原因的补充信息。

使用工作流通常如下：
(1) 选择要进行发现或验证的原语及所需算法；(2) 指定要处理的数据集和所需参数；(3) 执行算法并获取结果；(4) 根据需要筛选和排序结果。

这些步骤在 Web 应用程序中清晰地分开，而在控制台版本中，算法所需的所有信息（步骤 1–2）都可以直接通过命令行参数设置。Web 应用程序最初是为非IT专业人士设计的，因此除了主要功能外，还应提供一些提升使用体验的特性。例如，显示所选数据集的片段、带有详尽使用示例的用户友好界面、可直接使用的流水线，以及显示任务当前执行状态的进度条。Desbordante 本身和 Web 应用程序都内置了一组数据集。它们主要有两个用途。第一，帮助用户了解使用特定基元可以获得哪些数据洞察。第二，让新手熟悉该工具的工作流。除了内置数据集外，Web 应用程序还允许用户上传自己的数据集。这些数据集构成用户的个人库，其工作流与内置数据集相同。

4.2 面向用户的方面

我们想介绍的第一个特性，是原语发现和验证过程的可调节性。每种原语发现或验证算法都有一系列不同的选项。这些选项可分为两组：通用参数和原语专属参数。通用参数是执行任何发现任务都需要设置的属性：数据集、其分隔符，以及用于指示数据集是否包含表头的布尔开关。接下来是原语专属参数，其中首要且最重要的是算法。对于某些基元，哪种算法性能最佳或多或少是已知的，但对于另一些基元则并非如此。此外，发现算法在“宽表”或“长表”上可能表现不佳。由于特定数据集的特点，它也可能崩溃，因为不同算法建立在不同原理之上，例如可能需要过多的内存。因此，我们决定提供多种可选算法，在某些情况下则提供所有算法。

这些算法各自拥有一组受支持的参数。首先，用户可以指定要验证或查找的内容，以及针对原语实例的筛选条件。其次，如果某种算法支持近似原语的发现（或验证），则可以指定允许违反的程度。对于发现算法，以某种方式限制搜索的“深度”也很有用。发现过程耗时较长，而且并非总是需要所有实例。最后，如果所选算法支持多线程，用户可以指定用于原语发现或验证的线程数。

另一个重要功能是为原语验证任务提供自定义界面。该任务会报告特定原语是否成立。然而，如果它不成立，用户就需要解释，并希望了解诸如“还差多少？”以及“是什么阻碍它成立？”等问题的答案。提供这类信息至关重要，因为它是关于正在探索的数据的重要知识。这些信息可以指出数据中的错误，并标出“有问题”的记录。因此，需要一个能够提供这些信息的界面。

图2展示了度量函数依赖 [22] 验证的控制台版本中此类屏幕的一个示例。它显示了左侧相同、但右侧不同的记录簇。“x”标记的是右侧值与其他记录相距过远的记录。它们与同一簇中任意点的距离都大于指定值，因此很可能是离群值。用户可以根据距离、索引、离群值数量等多种参数，对簇以及簇内的记录进行排序。

原语发现任务还实现了结果界面，提供丰富的交互工具，可按各种参数排序、使用正则表达式筛选等。

4.3 流水线

除了原语发现和原语验证任务之外，Desbordante 还提供流水线。流水线是一套开箱即用的功能，基于已发现的原语实现，旨在方便非专业终端用户使用。尽管已发现的原语实例本身就很有用，但我们认为，展示如何利用它们完成实际工作也很重要。

Desbordante 中有两种类型的流水线：内置流水线和自定义流水线。前者存在于 Web 应用程序中，并提供丰富的交互界面。作为演示，我们在部署版本中加入了拼写错误检测流水线。其工作思路如下：

（1）找出几乎成立的函数依赖，即近似依赖 [23] 成立，但精确依赖并不成立。将这些依赖呈现给用户进行检查。（2）然后，对于用户选定的依赖，展示其簇——左侧相同而右侧取值不同的行组。这些行集合会导致精确函数依赖无法成立。在此界面中，用户可以检查右侧的差异，并判断是否存在拼写错误。解决特定簇中的冲突后，用户可以重新上传新版数据集并继续进行数据清洗。（3）为了减少展示的簇数量，用户界面提供了多个参数，用于筛选簇。用户还可以设置将依赖视为“几乎”成立的阈值。

请注意，命令行界面不提供内置流水线，因为它们需要大量交互。

内置流水线虽然适合作为演示，但实现起来需要付出大量工作。由于资源有限，我们只在网页版提供最实用的场景。同时，我们希望用户能够试用并构建自己的流水线。为此，我们提供了构建自定义流水线的功能。

如今的数据科学家使用 Python，因此，支持从 Python 程序调用原语发现和验证任务至关重要。为此，我们采用了 pybind11 [21] 库，提供必要的运算符和数据结构。借助这些绑定，用户可以调用 Desbordante 算法进行试验并构建自己的流水线。我们计划将常用的流水线添加到网页版，并为其开发易于使用的界面。

4.4 性能

与所有现有的开源解决方案不同，Desbordante 的发现部分完全采用现代 C++ 实现。Python 和 Java 等流行语言相对简单，因此开发速度较快，但它们也存在若干同样显著的缺点：

(1) 在投入相同编码精力的情况下，Java/Python 应用程序的性能通常不如 C++。(2) Java 应用程序的性能可能难以预测。由于 Java 不支持显式内存管理，程序依赖自动垃圾回收器，而垃圾回收器可能随时启动。因此，即使是单线程程序，连续调用的运行时间也可能存在显著差异。(3) Java 程序通常比 C++ 占用更多内存。

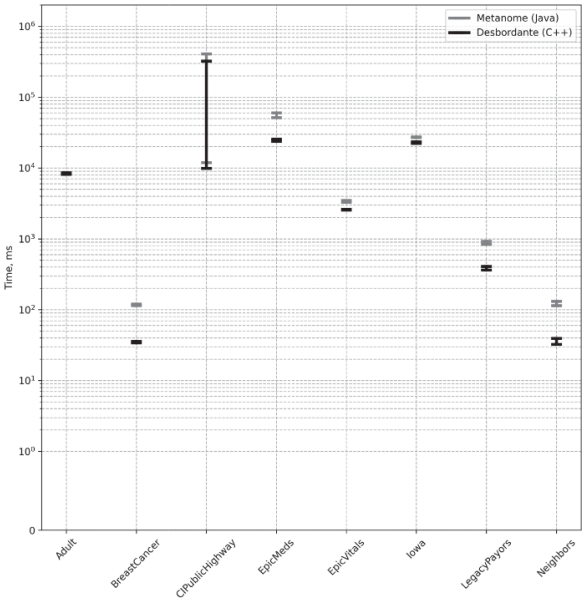

图 3：使用 Pyro 时 Desbordante 与 Metanome 的性能比较

(4) 最后，这些语言限制了低级优化的空间，例如通过 SIMD 指令实现向量化。这是解决高性能计算任务时的一项重大缺陷。

为证明我们的论点成立，我们对 Desbordante 与 Metanome [34]进行了实验性比较。为此，我们选择了 Pyro 算法 [23]，因为它是最适用于目标应用场景的潜力基元之一。该算法能够发现近似函数依赖。

结果如图 3 所示。获得的性能提升幅度为 1.19 至 3.43 倍，平均为 2.12 倍。虽然这些数值并不算特别高，但对于这样一个计算开销巨大的问题而言，仍是一项重要成果。

另一项显著优势是内存消耗的降低——Desbordante 的内存占用约为 Metanome 的一半。这一点至关重要，因为许多原语发现算法都受内存限制 [28]。因此，降低内存占用能够处理更大的数据集。

需要指出的是，我们尚未充分挖掘 C++ 实现的调优潜力。我们没有采用复杂技术（例如向量化），没有进行源代码性能分析，使用的也是标准数据结构和库，等等。目前，Desbordante 使用默认的 C++ 和 Boost 数据结构，我们也没有调整它们的参数。Desbordante 不依赖自定义内存管理库（分配器），而是使用 C++ 默认配置。众所周知 [25]，使用专用分配器是提升 C++ 程序性能的一种简单而高效的方法。因此，性能还有进一步提升的空间。

最后，我们还必须讨论依赖于分发 Java/Python 代码的方法。首先，我们认为这些方法不会带来太大改善。基元发现问题通常难以扩展，而朴素方法完全不可行（例如，参见 [31] 中的图 1）。原因在于，要得出答案，必须遍历整个数据集（或其中很大一部分）

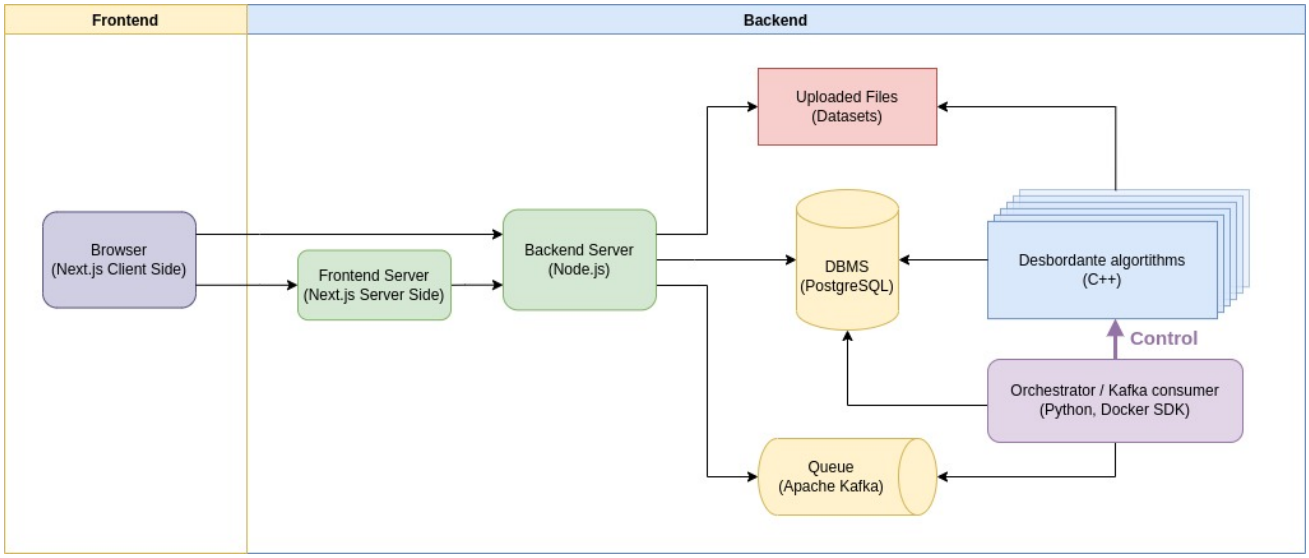

图4：Desbordante 架构详解

因此，我们认为充分发挥单节点处理的性能至关重要。

其次，由于分布式方法通常由数据洗牌方案和一些局部算法组成，因此可以说，基于分布式的方法并非与我们竞争，而是与我们互为补充。

4.5 支持的基元

由于第3节所述的原因，Desbordante 支持的基元集合与 Metanome 略有不同。目前，Desbordante 支持对以下基元进行发现和验证：

(1) 精确函数依赖的发现。我们支持 Metanome 团队实现的所有算法 [27]，包括 HyFD [28] 和近似算法 AID-FD [8]。(2) 使用 Pyro [23] 和 TANE [18] 算法发现近似函数依赖。(3) 使用 CTANE [15, 16] 算法及其变体发现条件函数依赖。(4) 使用 Spider 算法 [5]发现一元和 n 元包含依赖。(5) 验证度量依赖 [22]（目前仅限控制台版本）。(6) 发现模糊代数约束 [10]（目前仅限控制台版本）。(7) 发现关联规则。这部分代码改编自 Christian Borgelt 的[3] 实现 [9]，因为其效率高（用于 R 包[4]）且经受了时间考验。遵循他的建议，我们仅选用了 ECLAT、FP-Growth 和 Apriori 算法。(8) 验证图依赖 [17]（目前仅限控制台版本）

(9) 朴素数据剖析。为了扩大用户群，我们还实现了朴素数据剖析，其中包括一些简单统计量，例如每列的最小值、最大值、缺失值数量等。

目前，我们缺少 Metanome 所支持的许多基元，例如唯一列组合、顺序依赖等。不过，我们也支持一些 Metanome 不具备的其他类型（例如度量函数依赖），而这些类型与我们的用例更相关。此外，据我们所知，对于某些基元类型，例如图依赖、度量函数依赖和代数约束，我们的实现是唯一公开可用的。最后，我们计划在未来大幅增加基元数量并迎头赶上——其中一些缺失的基元已经在开发中。

5 架构

本节将介绍我们围绕核心系统 Desbordante 构建的 Web 应用程序架构。起初，Desbordante 是一个简单的控制台应用程序，通过命令行参数和标准输出作为用户界面。2021年夏季，我们形成了引言部分所述的愿景，并决定为其提供 Web界面。

在考虑其实现时，我们为该 Web 应用程序制定了以下要求：

（1）功能性。系统应能够并行执行多项用户任务。（2）可恢复性。系统应能够在发生各种意外错误时自行恢复。（3）效率和可管理性。系统应能够限制分配给特定用户甚至特定任务的计算资源。这项要求对于防止资源过度使用至关重要，同时也有助于实现资源调度。（4）可扩展性。系统架构应支持使用多个计算节点来执行用户任务。

因此，我们决定采用微服务架构，其中每个独立服务都执行一组特定任务。我们为处理用户请求、管理容器、执行任务、数据库和任务队列分别设置了服务。此外，微服务架构也很适合容器化。

[3] https://borgelt.net/fpm.html [4] https://www.rdocumentation.org/packages/arules/versions/1.6-8

展示微服务之间连接关系的整体架构见图4。

这种方法也简化了依赖管理，因为每项服务的容器中都预先安装了所有依赖项。因此，可以快速重新部署应用程序的新版本，尽早为用户提供新功能。

因此，我们构建了一个容错应用程序，这意味着即使其中一个微服务发生故障，应用程序仍会继续运行，并迅速重启发生故障的微服务。我们的架构具有很强的可扩展性，必要时可以启动更多任务执行器。

下面详细介绍这一架构。

（1）提供网页服务时，前端服务器会直接与用户交互。它负责服务器端渲染，这是一种将计算从客户端浏览器转移到服务器的技术。这能加快页面加载速度，并提升用户界面的响应能力。

（2）Node js Web服务器提供 API，可用于运行任务、发送文件和接收状态更新。每个任务到达时，服务器都会在数据库和队列中分别创建条目。Web客户端会定期向服务器发送 ping，服务器则会回复任务的进度信息。任务执行完成后，服务器会返回结果或错误信息（如果计算出错，或完成任务所需的计算资源过多）。

(3) 除 Docker 外，我们还开发了自己的容器编排器，用于管理执行器容器。该编排器也充当队列的消费者。对于每项任务，它都会创建一个新的执行器容器，并向其提供任务负载。此服务使我们能够限制执行器使用的计算资源。此外，如果执行器发生故障，它会将错误消息写入数据库。

(4) 我们使用 Kafka 作为任务队列。

这样便可以在多个编排器服务实例之间实现负载均衡。

(5) 在我们的用户存储当前可在运行或最近已完成任务的相关信息，例如进度信息、错误消息和计算结果。此外，它还包含用户档案、会话信息和文件元数据

(6) 应用程序的核心是其任务执行器，它将 Desbordante 用作库。其用途是在提供的数据上运行指定的算法。启动时，它会从数据库获取数据，然后开始执行。在计算进行期间，执行器会在数据库中更新任务状态。执行成功或失败时，它分别返回计算结果或错误信息。多个执行器容器可以同时运行，因此能够同时为多个用户提供服务。

微服务主体配有监控系统。该系统使用两种专用工具：Prometheus 和 Grafana。Prometheus 会从各项服务收集指标，定期从指定端点采集数据。Grafana 则以信息丰富的仪表板形式呈现这些数据，也能够在发生符合预定义条件的事件时发出通知。

收集到的数据能够帮助了解系统的运行状况、资源消耗和执行错误。这对于解决可能出现的硬件和软件问题至关重要。

6 未来计划

Desbordante 目前正在积极开发中，主要有两个方向：改善用户体验（广义而言）和添加新的原语。前者包括以下任务：

- 数据上传的替代方式：通过外部文件链接、数据库连接器，或导入序列化数据结构（例如 pandas 腌制数据框或 NumPy 数组）；
- 以最常见的数据交换格式导出结果；
- 网络 API 令牌，以便使用远程高性能服务器执行发现任务；
- 数据操作，例如原位表编辑、列重命名，或根据用户定义的公式创建新列；
- 针对数据集单元格的正则表达式搜索；
- 扩展用户界面和管理员仪表板。

第二项涉及扩充可用原语的种类。我们计划从两个方向着手：首先，通过添加缺失的原语来追赶 Metanome；同时，我们也计划继续践行自己的理念，让鲜为人知的原语得到关注。近期计划包括实现以下原语：

- 匹配依赖
- 顺序依赖
- 否定约束
- 差异依赖
- 唯一列组合
- 各种宽松函数依赖
- 图依赖发现
- 高级数据集统计信息

最后，我们还计划探讨系统方面的问题：

- 设计智能的结果缓存，并检查结果是否完整。
- 实现流处理和基元的动态重新计算。

致谢

我们谨向 Nikita Talalay 和 Bulat Biktagirov 致谢，感谢他们对本项目的贡献。我们还要感谢 Anna Smirnova 在本文准备过程中给予的帮助。

7 结论

本文介绍了 Desbordante——一款开源数据分析器，专注于发现科学密集型数据模式。Desbordante 旨在让更广泛的公众能够使用工业级基元发现，重点面向并非 IT 专业人员的领域专家。与类似系统不同，它的构建着重考虑了多用户环境下的工业应用。它高效、具有抗崩溃能力且可扩展。其效率通过使用 C++ 实现发现算法来保障；其抗崩溃能力得益于广泛采用容器化；其可扩展性则基于容器复制。

参考文献

[1] James Campbell Abe Gong。 [无日期]。Great Expectations。https://github.com/great-expectations/great_expectations。[2] Ziawasch Abedjan、Lukasz Golab 和 Felix Naumann。2015年。《关系数据剖析：综述》。《*The VLDB Journal*》24卷4期（2015年8月），557–581。https://doi.org/10.1007/s00778-015-0389-y [3] Ziawasch Abedjan、Lukasz Golab、Felix Naumann 和 Thorsten Papenbrock。2018年。《数据剖析》。Morgan & Claypool Publishers。

[4] Charu C. Aggarwal 和 韩家炜。2014年。《频繁模式挖掘》。施普林格出版公司。[5] Jana Bauckmann、Ulf Leser、Felix Naumann 和 Veronique Tietz。2007年。高效检测包含依赖。载于《2007年IEEE第23届国际数据工程会议论文集》。1448–1450。https://doi.org/10.1109/ICDE. 2007.369032[6] Leopoldo Bertossi、Loreto Bravo、Enrico Franconi 和 Andrei Lopatenko。2008年。在完整性约束下修复数据库中的数值属性的复杂性与近似。Information Systems 33卷4期（2008年），407–434。https://doi.org/10.1016/j.is.2008.01.005 选自第十届数据库编程语言国际研讨会论文集（DBPL 2005）。[7] George Beskales、Ihab F. Ilyas 和 Lukasz Golab。2010年。在硬约束下对函数依赖违反的修复进行抽样。VLDB基金会会刊 3卷1–2期（2010年9月），197–207。https://doi.org/10.14778/1920841.1920870[8] Tobias Bleifuß、Susanne Bülow、Johannes Frohnhofen、Julian Risch、Georg Wiese、Sebastian Kruse、Thorsten Papenbrock 和 Felix Naumann。2016年。大型数据集的函数依赖近似发现。载于《第25届ACM信息与知识管理国际会议论文集》（CIKM '16）。计算机协会，纽约，美国，1803 1812 10 1145 2983323 2983781 –。[9] 2005 https://doi.org/ . / . Christian Borgelt。无日期。FP-Growth算法的实现。载于《第一届开源数据挖掘国际研讨会：频繁模式挖掘实现论文集》（OSDM '05）。计算机协会，纽约，美国，1 5 10 1145 1133905 1133907 –。[10] 2003 https://doi.org/ . / . Paul G. Brown 和 Peter J. Hass。无日期。BHUNT：自动发现关系数据中的模糊代数约束。载于《第29届超大型数据库国际会议论文集——第29卷》（VLDB '03）。VLDB基金会，668–679。[11] Simon Brugman。2019年。pandas-profiling：用于 Python 的探索性数据分析。https://github.com/pandas-profiling/pandas-profiling。[12] Loredana Caruccio、Vincenzo Deufemia 和 Giuseppe Polese。2016年。宽松函数依赖——方法综述。《IEEE知识与数据工程汇刊》28卷1期（2016年），147–165。https://doi.org/10.1109/TKDE.2015.2472010[13] Xu Chu、Ihab F. Ilyas 和 Paolo Papotti。2013年。整体数据清洗：将违反置于上下文中。载于《2013年IEEE第29届国际数据工程会议》（ICDE）。458 469 10 1109 2013 6544847 –。[14] https://doi.org/ . /ICDE. . Xu Chu、John Morcos、Ihab F. Ilyas、Mourad Ouzzani、Paolo Papotti、Nan Tang 和 Yin Ye。2015年。KATARA：利用知识库和众包实现可靠的数据清洗。VLDB基金会会刊 8卷12期（2015年8月），1952–1955。https://doi.org/10 14778 2824032 2824109 . / . 范文飞、Floris Geerts、Laks V. S. [15] Lakshmanan 和 Ming Xiong。2009年。发现条件函数依赖。载于《第25届国际数据工程会议论文集（ICDE 2009）》，2009年3月29日至4月2日，中国上海。电气电子工程师学会，1231–1234。https://doi.org/10.1109/ICDE.2009. 208[16] 范文飞、Floris Geerts、李建中和 Ming Xiong。2011年。发现条件函数依赖。《IEEE Trans. Knowl. Data Eng.》23卷5期（2011 683 698 10 1109 2010 154 ），–。[17] 2016 https://doi.org/ . /TKDE. . 范文飞、Yinghui Wu 和 Jingbo Xu。无日期。图的函数依赖。载于《2016年数据管理国际会议论文集》（SIGMOD '16）。计算机协会，纽约，美国，1843 1857 10 1145 2882903 2915232 –。https://doi.org/ . / . Y. [18] Huhtala、J. Kärkkäinen、P. Porkka 和 H. Toivonen。1999年。Tane：一种高效发现函数依赖和近似依赖的算法。《计算机杂志》42卷2期（1999年），100–111。https://doi.org/10.1093/comjnl/42.2.100[19] Madelon Hulsebos、Kevin Hu、Michiel Bakker、Emanuel Zgraggen、Arvind Satyanarayan、Tim Kraska、Çagatay Demiralp 和 César Hidalgo。2019年。Sherlock：一种基于深度学习的语义数据类型检测方法。载于《第25届ACM SIGKDD知识发现与数据挖掘国际会议论文集》（KDD '19）。计算机协会，纽约，美国，1500 1508 10 1145 3292500 3330993 –。https://doi.org/ . / . [20] 2019 Ihab F. Ilyas 和 Xu Chu。无日期。数据清洗。计算机协会，纽约，美国。[21] Wenzel Jakob、Jason Rhinelander 和 Dean Moldovan。2017年。pybind11——C++11 与 Python 之间的无缝互操作。https://github.com/pybind/pybind11。[22] Nick Koudas、Avishek Saha、Divesh Srivastava 和 Suresh Venkatasubramanian。2009年。度量函数依赖。载于《2009年IEEE第25届国际数据工程会议》。1275–1278。https://doi.org/10.1109/ICDE. 2009.219[23] Sebastian Kruse 和 Felix Naumann。2018年。高效发现近似依赖。VLDB基金会会刊 11卷7期（2018年3月），759–772。https://doi. org/ 10 14778 3192965 3192968 . / . Heiko Müller、Sonia Castelo、Munaf Qazi 和 [24] Juliana Freire。2021年。从论文到实践：OpenClean开源数据清洗库。VLDB基金会会刊 14, 12 2021 2763 2766 10 14778 3476311 3476339 （10月），–。[25] 2018 2 https://doi.org/ . / . “No Bugs” Hare。无日期。测试内存分配器：在尝试模拟真实世界负载时比较 ptmalloc、tcmalloc、hoard 与 jemalloc。http://ithare.com/testing-memory-allocators-ptmalloc2-tcmalloc-hoard-jemalloc-while-trying-to-simulate-real-world-loads/[26] Thorsten Papenbrock、Tanja Bergmann、Moritz Finke、Jakob Zwiener 和 Felix Naumann。2015年8月12日。2015年，1860–1863，10，14778，2824032，2824086。使用 Metanome 进行数据剖析。VLDB基金会会刊（8月），–。https://doi.org/ . / .

[27] Thorsten Papenbrock、Jens Ehrlich、Jannik Marten、Tommy Neubert、Jan- Peer Rudolph、Martin Schönberg、Jakob Zwiener 和 Felix Naumann。2015年。函数依赖发现：七种算法的实验评估。*VLDB*基金会会刊 8, 10（2015年6月），1082–1093。https://doi.org/ 10.14778/2794367.2794377[28] Thorsten Papenbrock 和 Felix Naumann。2016年。一种函数依赖发现的混合方法。载于*2016*年数据管理国际会议论文集（*SIGMOD '16*）。计算机协会，纽约，NY，美国，821–833。https://doi.org/10.1145/2882903. 2915203[29] Theodoros Rekatsinas、Xu Chu、Ihab F. Ilyas 和 Christopher Ré。2017年。Holo-Clean：利用概率推断进行整体数据修复。*VLDB*基金会会刊 10, 11 2017 1190 1201 10 14778 3137628 3137631 （8月），–。[30] https://doi.org/。/。El Kindi Rezig、Mourad Ouzzani、Walid G. Aref、Ahmed K. Elmagarmid、Ahmed R. Mahmood 和 Michael Stonebraker。2021年。Horizon：可扩展的依赖驱动数据清洗。*VLDB*基金会会刊 14, 11（2021年10月），2546 2554 10 14778 3476249 3476301 –。https://doi.org/。/。[31] 2019 Hemant Saxena、Lukasz Golab 和 Ihab F. Ilyas。分布式依赖发现算法实现。*VLDB*基金会会刊 12, 11（2019 1624 1636 10 14778 3342263 3342638 年7月），–。[32] 2021 https://doi.org/。/。Jie Song 和 Yeye He。Auto-Validate：利用从数据湖中推断出的数据域模式进行无监督数据验证。载于*2021*年数据管理国际会议论文集（*SIGMOD '21*）。计算机协会，纽约，NY，美国，1678–1691。https://doi.org/10.1145/3448016.3457250[33] Shaoxu Song、Fei Gao、Ruihong Huang 和 Chaokun Wang。2022年。面向多样性和真实性扩展的数据依赖：族谱。*IEEE*知识与数据工程汇刊 34, 10（2022年），4717–4736。https://doi.org/10 1109 2020 3046443 . /TKDE.。Maxim Strutovskiy、[34] Nikita Bobrov、Kirill Smirnov 和 George Chernishev。2021年。Desbordante：探索依赖发现算法极限的框架。载于*2021*年开放创新协会第*29*届会议（*FRUCT*）。344–354。https://doi.org/10.23919/FRUCT52173.2021.9435469[35] Petr Tsurinov、Oleg Shpynov、Nina Lukashina、Daria Likholetova 和 Maxim Artyomov。2021年。FARM：层次关联规则挖掘与可视化方法。载于第*12*届*ACM*生物信息学、计算生物学与健康信息学会议论文集（*BCB '21*）。计算机协会，纽约，NY，美国，文章70，1页。https://doi.org/10 1145 3459930 3469499 . /。Mohamed Yakout、Laure [36] Berti-Équille 和 Ahmed K. Elmagarmid。2013年。别害怕SCARED：采用最大似然和有界更改进行可扩展自动修复。载于*2013*年*ACM SIGMOD*数据管理国际会议论文集（*SIGMOD '13*）。计算机协会，纽约，NY，美国，553–564。https://doi.org/10.1145/2463676. 2463706[37] C.T. Yu 和 W. Sun。1989年。用于语义查询优化的自动知识获取与维护。*IEEE*知识与数据工程汇刊 1, 3（1989年），362–375。https://doi.org/10.1109/69.87981