从数据库中提取语法模式

Andrew Ilyas, Joana M. F. da Trindade, Raul Castro Fernandez, Samuel Madden

摘要

许多数据库列包含符合模式的字符串或数字数据，例如电话号码，日期，地址，产品标识符和员工 ID 等等。这些模式在许多数据处理应用程序中很有用，包括了解字段名称不明确时特定字段表示什么，识别异常值以及在数据集中查找相似的字段。一种表示这种模式的方法是学习数据库中每个字段的正则表达式。不幸的是，现有的技术在学习常规的表达式上的速度很慢，只有几千个值的列就需要花费数百秒的时间。相比之下，本文开发了 XSYSTEM，这是一种高效的方法，可以在很短的时间内学习数据库列上的模式。XSYSTEM 方法可以在很短的时间内学习数据库列上的模式。这些模式不仅可以快速构建，而且可以捕获许多关键应用程序，包括检测异常值，测量列相似性以及为列分配语义标签（基于正则表达式库）。

关键词：模式学习；异常检测；相似性查询；示例编程

1 引言

现代企业将数据存储在各种不同的系统中，包括事务性 DBMS，数据仓库，数据湖，电子表格和平面文件。数据分析师经常需要合并来自这些不同数据集的数据，并经常合并来自甚至更多来源的外部数据。在这种情况下的一个关键挑战是找到可以组合起来以回答某些关键问题的相关数据集。

由于 ID 格式的多样性，简单的文本搜索不足以找到相关的表和属性名称。实际上，由于这些标识符不具有可比性，因此我们甚至无法执行近似搜索来找到相似的内容。手动在不同格式的标识符之间建立映射并创建查找表是一种昂贵的选择，更好的选择是使用有用的元数据标记相关属性。但是，在拥有大量数据的公司中，手动标记也是不可行的。因为它需要大量时间，并且容易出错，因为它可能会丢失许多包含相关信息的表，尤其是在考虑外部数据时。

为了解决此问题，我们注意到企业数据库中的许多相关属性都是高度结构化的，即它们遵循简单的语法模式。结构化属性的比较常见的示例是日期、产品标识符、电话号码、枚举类型（性别等），通常这些列作为字符串存储在数据库中，但是如果它们可以用关于值的格式的更丰富的结构信息来标记，则可以更有效地进行索引、搜索和比较值以及查找异常值。

本文提出了一种从数据集中提取语法模式到称为 XTRUCTURES 的数据结构中的方法 XSYSTEM。XTRUCTURE 代表一种语法模式，可以与其他 XTRUCTURE 以及正则表达式进行比较。一旦 XSYSTEM 学习了一组模式，分析人员就可以使用它们来执行几个通常执行的任务，包括：自动标签分配，其中通过将数据项与已知类库（编写为 regex 或 XTRUCTURES）进行比较来为数据项分配一个类; 查找语法上相似的内容，其中比较所学习的 XTRUCTURE 以查看它们是否相似，以及异常检测，其中将单个项目的所学习的 XTRUCTURE 与其它 XTRUCTURE 进行比较以检查其结构是否不同。

在我们生活的世界中，每天我们都有大量的数据以高速度和大容量的形式生成，在标准的大数据场景中，学习速度对于现实世界的场景至关重要，因此本文所提出的方法在处理数据库中的数据时是具有非常重要的现实意义的。

2 相关工作

在本节中将会介绍与 XSYSTEM 相关的若干技术和其研究的贡献。

2.1 信息提取

XSYSTEM 与信息提取（IE）相关，因为它从数据中提取结构表示。大多数 IE 技术从完全非结构化的数据（如文本）或半结构化数据（如 XML 和 HTML）中提取结构。此外，这些技术中的大多数需要可变的人力投入。XSYSTEM 必须自动工作，它对结构化数据进行操作，生成一个简洁的模式，表示输入字符串集合的语法结构。XSYSTEM 补充了 IE 中的现有技术，并在大型企业的重要应用中取得了良好的性能。

2.2 正则表达式推断

这些技术从字符串集合中提取语法模式。最近的工作主要是使用多目标优化和积极的空间修剪来减少推理过程的运行时间[1]。对于企业设置中使用的方法来说，性能仍然是一个问题，因为它们的评估显示，学习具有 500 个条目和 32 个线程的数据集需要 40 分钟以上。XSYSTEM 减少了正则表达式的不必要的表达性，以提高性能。

其他方法可分为是否需要反面例子。那些需要反面例子的人很少适合企业环境。在只需要正面例子的系统中,[2]、[3]和[4]是最相关的。在[2]中，将输入字符作为其字符类（在这种情况下称为令牌类）来处理，以生成更高级别的输入数据抽象。他们的方法学习循环 DFA，而在本文中展示这种表达能力对于本文的目标应用程序来说是不必要的。

在 XML 的上下文中,[5]中的方法从几个积极的例子中学习了简洁的正则表达式，并且还可以从该方法学习的表示中生成可读字符串，从而为比较打开了一条道路。最后，ReLIE[3]需要示例正则表达式，然后进一步细化。本文的不同之处在于，本文的操作没有人工输入。与所有这些工作不同，本文的工作专注于：第一，设计 XSYSTEM 以捕获数据库中的语法模式，而不是解决为无限语言学习正则表达式这一普遍且更复杂的问题；第二，支持学习模式的有效比较，这有助于识别语法相似的内容、自动标记数据和识别语法异常值。

2.3 程序合成

基于程序综合的方法已经得到了广泛的应用[6]-[7]。与 XSYSTEM 不同，他们的目标通常是操作和转换数据，例如用于数据清理。这意味着他们需要在内部构建和维护的结构化的复杂性要高于 XSYSTEM。例如，BlinkFill[8]必须构建一个 InputDataGraph 来转换数据，这比构建 XSYSTEM 要昂贵得多，而且对我们的目标来说是不必要的。

3 本文方法

3.1 本文方法概述

首先，根据本文的描述，定义一个 Xtructure 结构的数据模型。模型结构图如图 1所示：

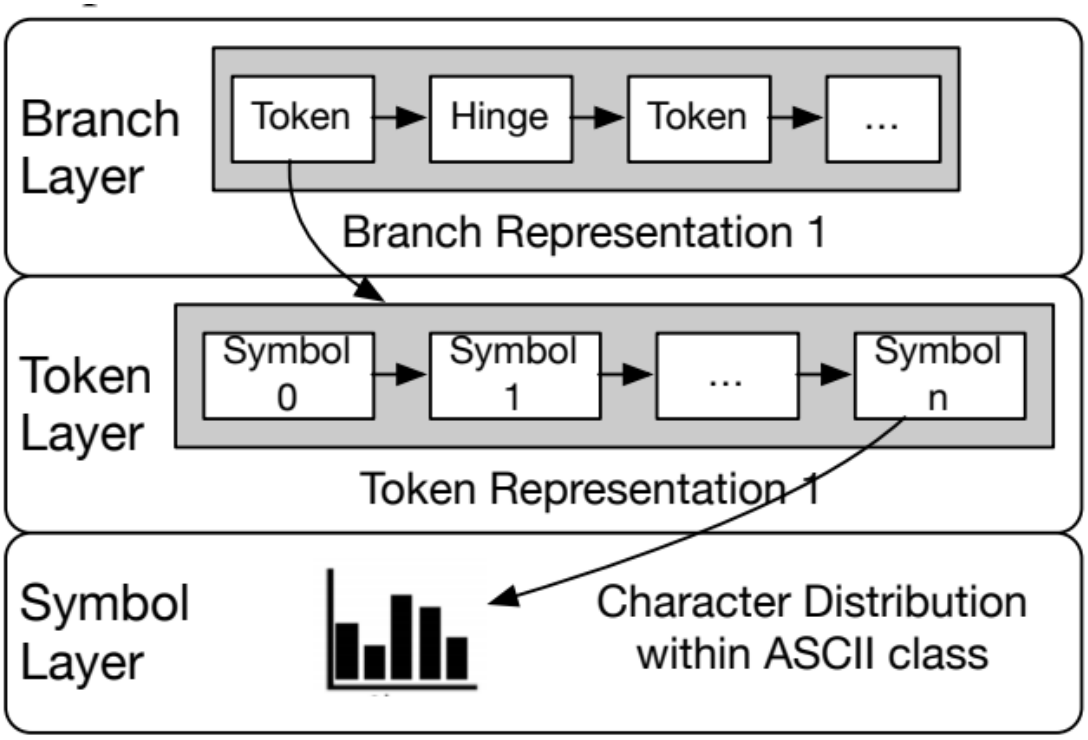


图 1: Xtructure 数据模型

XSYSTEM 的目标是从示例中增量地（一个元组接一个元组地）学习 XTRUCTURE。为了做到这一点，首先设计了一个架构，它允许我们对每个示例进行概率建模，从而在任何时候输出“当前”表示。XTRUCTURE 的结构有分为三个层次，通过以定义良好的方式将元组传递到这个分层结构中，我们可以将元组拟合到模型中。这些层是分层组织的，每个层都负责学习过程的不同方面。接下来我们将解释每个层的作用。

层次结构中的底层是符号层，它保存输入元组中给定位置处出现的 ASCII 字符的分布。这允许我们将元组中的位置表示为基于该分布的字符类、语句、单个字符或通配符（“.”）。例如，如果将一系列 mm/dd/（yy）yy 日期输入到 XTRUCTURE 中，则第一个字符将包含一年中仅包含值 1 或 0（因为 0 ≤months≤ 12）的分布。第二个将最终收敛到 [0，9] 上的均匀分布，因此它将用字符类数字 D 表示。

令牌层表示来自原始元组或令牌的字符序列，所述字符序列是通过根据一组定界符分割原始元组而获得的，例如, “-”、“/” 等等。一般来说，分隔符通常捕获元组的子结构。以“10/1/2017”日期作为一个例子，在这里三个令牌由/分隔。每个标记都以标记表示形式表示，标记表示形式只是符号的链接列表（来自符号层）。对于日期，日期中的“月份”将是标记层中的标记，最终表示为（0-1）D。当数据中没有可用的分隔符时，整个字符串表示为单个标记。

不同长度的令牌不能用单个令牌层表示。层次结构中的下一层称为分支层，用于处理可变长度数据。分支层由令牌层的列表组成，并且可以表示整个元组。特别地，分支层表示与分隔符交织的令牌（由令牌层捕获）的列表。在我们的“日期”示例中，我们可能会发现日期具有两种不同的年份格式，即 4 位数和 2 位数。这两个变体将在分支表示中用两个不同的分支来表示。每个 XTRUCTURE 都有几个分支来表示具有不同语法模式的属性，例如，具有不同长度的元组。由于数据质量问题，以及 ID、大写拼写错误等原因，查找具有许多不同格式的日期是很常见的。

XTRUCTURE 在每个输入元组被消耗之后被适配。当 XSYSTEM 得到一个新的元组时，它会为元组选择一个现有的分支（如果存在的话），或者创建一个新的分支并将输入作为种子。该决定是基

于评分拟合的测量做出的。然后，分支表示基于一组分隔符将输入分割成标记 K，并将每个标记 k 拆分成字符，从而更新标记和符号层。

接下来的将会大概介绍如何实现学习 Xtructure 的。包括得分拟合、分支合并算法、对输入元组进行标记化并将字符馈送到各个层的方法。

3.2 评分拟合元组

在学习 XTRUCTURE X 时，我们必须了解元组 t“适合”X 定义的结构的程度。为此，我们引入了一种评分拟合度量。更正式地，给定元组 t 和 XTRUCTURE，X，我们定义操作 d（t，X）：R+，表示 t 偏离 X 所代表的模式有多远。这个函数对于拟合新的示例以及将 XTRUCTURE 与其自身以及与用其他方法学习的表示进行比较非常有用。为了构建这个函数，我们不是将 t 中的每个字符与表示 X 中对应的“字符”进行比较（这是不明确的，因为我们的模型拥有字符而不是单个字符的分布），而是查看符号层所表示的字符 S，并为每个 s_i 与符号层 l_i 中的表示的匹配程度分配得分 d($S_i \in S$,l_i)。为了定义 d，我们使用 GET-ASCII-CLASS（c）作为 UNIX 类（例如，字母数字、空白等）以及 l.class 作为符号层 l 的字符类。然后将 l.is_class 定义为布尔值，表示层的表示是否是其字符类。该决定基于卡方检验。如果其 p 值不能表示为字符的“OR”运算，则使用如下公式：

$$d(s_i, l_i) = \begin{cases} 1, \text{if GET-ASCII-CLASS}(s_i) \neq l_i.class \\ \alpha, \text{if not } l_i.\text{is\_class and } s_i \notin l_i.chars \\ 0, otherwise \end{cases}$$

其中，$l_i.chars$ 是符号层 l_i 中表示的字符，并且参数 α 用于确定与仅在类中的匹配（即两个字符）相比有多少精确字符匹配被优先化。在此设定 α = 15 作为相对权重的合理取值。

我们使用 d 将符号层的评分拟合传播到 XTRUCTURE 的各个层，从而得到元组相对于模型的一般评分拟合。具体地，对于令牌表示 K、分支表示 B 和模型化表示 X，元组 t 到 X 的距离定义为：

$$d(t, X) = \min_{b \in B} d(t, b)$$

即，元组与 X 的分支表示之一 b 的最小距离，其又被定义为：

$$d(t, b) = \sum_{k_i \in b, t_i \in t} d\left(t_i, k_i\right)$$

其中 t_i 是输入元组 t 的标记，其与 X 的标记结构 k_i 比较如下：

$$d\left(t_i, k\right) = \left[\sum_{l_i \in k, s_i \in t_i} d\left(s_i, l_i\right) \right] + |\text{len}\left(t_i\right) - \text{len}(k)|$$

注意，最后一个等式中的额外项用于填充空字符，取令牌结构 X 中的 k_i 和元组 t 中的标记 t_i 之间较短的一个。这确保了 XSYSTEM 不会错误地惩罚底层有限语言的较小的有效实例，同时仍然在结构中为它们创建一个新的分支。例如，一列包含几个“123”实例和一个“1”实例，后者将用 2 个空字符填充。

3.3 表示多个分支的算法

实际上，来自同一属性的数据可能包含具有不同语法模式的值。例如，ID 可能是一个 10 位数字，或者简单地为“N/A”。这一现象启发了 XTRUCTURE 的多重分支表示（也就是为什么我们允许 R 是 b– 维的）。然而，我们无法事先知道一组示例中有多少不同的模式，XTRUCTURE 必须以某种方式管

理多个分支，适当地更新和表示它们。

给定一个新的输入元组，XSYSTEM 必须决定是将其放入现有的 XTRUCTURE 分支，还是创建一个新分支来捕获元组的语法结构。为此，我们使用得分拟合。对于每个输入 t，XSYSTEM 通过执行 $b_{\text{best}} = \arg\min_b d(t,b)$ 来找到“最佳匹配”分支; 如果 d(t,b_{best}) 低于分支阈值，则元组适合于该分支，否则创建新的“空”分支。这个分支阈值的存在给如何调整它带来了挑战，为了避免人工调整这个超参数，所以提出了一种自适应分支合并技术。该算法的详细实现将会在复现细节中进行描述。

3.4 标记化和字符拟合

为了更新令牌层，输入元组被分割成令牌，然后每个令牌被馈送到其对应令牌结构的层。标记器使用特殊字符（分隔符）作为参考对齐。这些字符的位置在一个字符串上通常是数据类型的指示符。例如，IPv4 地址块以“.”分隔，而日期通常用“/”或“-”分隔。

3.5 建模表示法

在建模期间，在接收到新示例并确定了令牌结构之后，每个令牌被馈送到其令牌结构的层。具体的实现方法也会在复现细节中进行描述。

3.6 元组生成和可读性

XTRUCTURE 需要生成元组，这些元组虽然不一定是给定示例的一部分，但符合示例的定义域（形式上是 f(X)）。这对于比较是必要的，我将在复现细节中解释如何从 XTRUCTURE 生成元组。与元组的生成相关的是人类可读的 XTRUCTURE 的字符串表示，这是向人类提供数据概览的有用属性，我也在复现细节中对此进行了描述。

4 复现细节

4.1 复现细节描述

本文的方法从示例中逐步学习语法。对于每个示例，利用已知实体中存在的分隔符，将提取模式的问题分解为学习由这些分隔符分隔的每个标记的语法。用于学习每个令牌的底层数据结构是分支线性分布序列，其等效于确定性非循环有限状态自动机（DAFSA），DAFSA 比经常用于正则表达式学习的最小确定性有限自动机（DFA）学习起来渐近地更简单。学习过程依赖于一个分支和合并策略，它允许我们逐渐地适应新的观察到的例子。这允许我们捕获出现在同一列中的不同语法结构。这种分支和合并策略也是 XSYSTEM 中使用的并行化方法的核心。

4.1.1 合并分支算法

算法主要流程如下，伪代码如算法 1 所示。伪代码中隐藏了一些不直观的依赖于数据的超参数，主要是一个表明最大分支数量的 maximum branches 参数，该参数表示 XTRUCTURE（形式定义中的 b）所要表示的结构的最大数量。该参数可以基于领域知识或用户偏好来设置，如果分析人员知道有 3 种方式来表示企业实体，他可以选择 3 作为分支机构的数量，不超过预期出现在数据中的分支机构数量。

给定一个固定的分支阈值和用户所需的最大分支数，XSYSTEM 将按如下方式进行：对于每个新输入，确定它是否“适合”任何现有分支（第 1-2）；如果是，则将其添加到该分支，否则，创建一个

新分支（第 4 行）。如果分支的数量超过了指定的最大值（第 6 行），我们计算分支之间的成对距离。两个最近的分支 b1 和 b2 通过将来自被包含分支的生成元组拟合到一个包含中而被合并（行 7），并且新的“分支阈值”被设置为 d（b_1，b_2）（行 7）。

这种自适应机制允许 XSYSTEM 校正未达到的初始阈值，但不能校正超过的初始阈值，因此在实践中，初始分支阈值被设置为一个小的 $\varepsilon > 0$。整个算法，包括挑选最佳分支和分支合并，在算法 1 中更详细地示出。

Procedure 1 Fitting new words into XTRUCTURE(learn new word).

Input: Int *branching_threshold,* String *word,* Int *max_branches*
Output: Int *result*

1 $best\_branch \leftarrow \arg\min_{\text{b}\in \text{ branches}}\ b.fit\_score(word)$
if *best_branch.fit_score(word)* < *branching_threshold* **then**
2 $best\_branch.add(word)$
3 **else**
4 $branches.add(newBranch(word))$
5 **end**
6 **if** *branches.length* > *max_branches* **then**
7 // fit(B_i,B_j) returns how well B_i fits into B_j
$B_{outer}, B_{inner} \leftarrow \arg\min_{(B_i,B_j)\in \text{ branches}}\ fit\,(B_i, B_j)$
$branching\_threshold \leftarrow fit(B_{outer}, B_{inner})$
$B_{\text{outer}}.add\_word(w) \forall w \in B_{\text{inner}}.learned\_words$
delete B_{inner}
8 **end**

4.1.2 拟合元组的符号层表示

在建模期间，在接收到新示例并确定了令牌结构之后，每个令牌被馈送到其令牌结构的层。如前所述，符号层保存它所看到的字符的分布，并在具有统计显著性时用它们的字符类来表示它们（参见算法 2）。在多数字符类中的每个字符具有相同可能性的假设下，将每一层建模为采样问题。然后进行卡方独立性检验，确认或拒绝该假设; 如果被确认，则该层通过其字符类来表示其自身（算法 2 中的第 14-15 行）。如果零假设被拒绝，则表示中应捕获的数据源中存在显著偏差，因此该层改为在 or 语句中按频率降序枚举所有拟合元组，直到捕获到分布的指定“捕获百分比”。这对应于算法 2 的第 17-20 行。每当遇到一个新的例子时运行这个过程，确保我们总是建模一个有效的 XTRUCTURE。

Procedure 2 Symbol layer representation of fitted tuples.

Input: String *all_chars_seen,* Int *capture_threshold*
Output: String *result*

$class\_proportions \leftarrow$ proportion of each character class seen
// e.x. ”A-Z”: 0.5, ”a-z”: 0.25, ”1-9”: 0.25
$max\_class \leftarrow argmax(class\_proportions)$
$max\_proportion \leftarrow class\_proportions[max\_class]$
if *max_proportion > 0.95* **then**
    $chars\_to\_capture \leftarrow filter(x \rightarrow x \in max\_class, all\_chars\_seen)$
    $histogram \leftarrow histogram(chars\_to\_capture, bins = size(max\_class))$
else
    $chars\_to\_capture \leftarrow all\_chars\_seen histogram \leftarrow histogram(chars\_to\_capture, bins = sum(size(class) \forall class \in class\_proportions))$
end
if *ChiSquared(histogram) > p* **then**
    return max_class
else
    $captured \leftarrow 0$
    sort all chars seen by frequency
    $representation \leftarrow []$
    while *captured < capture_threshold* **do**
        $next\_char \leftarrow all\_chars\_seen.next()$
        $representation.add(next\_char)$
        $captured \leftarrow captured + frequency(next\_char)$
    end
    return “|”.join(representation)
end

本算法中主要的参数有：

捕获阈值：随着捕获阈值的增加，则在 XTRUCTURE 中包含更多的数据分布，代价是可能包含离群值; 在实验中，如果捕获阈值是 0.9，则在单个错误示例上训练之后，数据结构将可能不会改变。

最大分支数：所采用的数据集对于最大分支参数是相当稳健的，但它仍然有影响。将其设置为 1 导致单个分支识别 DD/MM/YY 形式的日期，这意味着不对 D/M/YY 和 NA 形式的日期进行评分，而将该参数设置为极高导致为印刷错误和异常值分配新分支，这也是不期望的。

4.1.3 优化

1. 并行学习

通过使用分支合并算法可以并行学习 XTRUCTURE。在拟合模型时，可以使用多个工作器，每个工作器阅读不相交的元组集并独立地拟合它们。这样做的好处是利用了现代体系结构中现成的并行性，但导致每个属性有多个表示。在这一点上，可以使用分支合并算法来合并不同构建模型的分支，从而得到一个等价于单个工作器所学习的表示。

2. 提前停止

当从结构化数据中学习时，通常大部分计算时间都用于拟合对最终 XTRUCTURE 没有贡献的元组。例如，一长串格式良好的日期，经过几个元组之后，我们建模的表示将反映模式，并且不会随着

处理更多元组而改变。

当模型已经收敛并且在消耗了一定数量的新元组之后不再改变时，我们可以停止学习。为此，在跟踪拟合过程中新元组的拟合变化了多少时，最初预计得分会有很大变化，随着时间的推移，得分会下降并变得稳定，特别是当数据是规则的时候。提前停止的过程是用到了的中心极限定理的一个应用所启发的。提前停止过程如算法 3 所示。当属性中的平均适合度得分元组降到阈值以下时，我们停止进行拟合。为了确保可信度，我的想法是对分布进行分组抽样，取样本平均值。这些样本平均值近似于 μ（所需平均值）附近的正态分布。因此，为了确定过程是否应该提前停止，我在实验中生成 n 个元组的组，并计算它们在学习的 XTRUCTURE 中的平均适应度，以及近似正态分布的标准差。这种技术允许我们跳过大量数据，同时仍然找到良好的近似表示。当属性具有许多不同的分支（仅在以后才能看到）时，该方法将失败。因此，默认情况下禁用此技术，当用户知道数据是高度规则的或随机混洗的时，应启用此技术。

Procedure 3 Fitting tuples to branches.

Input: String *word*
Output: Int *result*

```
all_scores ← []
 latest_scores ← []
 if not done_adding then
    score ← ∑_{1⩽i⩽| layers |} layers[i].add_and_output_score(word[i])
     latest_scores.append(score)
     if len(all_scores)=30 then
        score← avg(latest_scores)
         all_scores.append(score)
         latest_scores← []
    end
    current_std(all_scores)
     if len(all_scores)>needed_sample_size(current_std) then
            doneadding ← true
    end
end
```

其中使用 95% 的预期置信度来估计样本量，如下所示。

Procedure 4 needed sample size.

Input: Float *current_std*
Output: Int *result*

```
return int ((1.96 ∗ x/0.1)^2) // this is a normal distribution
```

4.1.4 从 XTRUCTURE 生成元组

为了生成元组，XSYSTEM 自底向上遍历 XTRUCTURE 的各个层。它通过其符号层生成字符。这些分隔符由令牌层连接成令牌，令牌层还负责根据需要对分隔符进行交织。最后，标记被连接成分支，生成器随机选择将被选择来生成输出元组的分支。

为了确保每个符号层生成的字符指向能够很好地表示结构的元组，每个符号层从其对应的字符分布中随机抽取，从而从符号层生成一个字符串，而不是返回其字符分布的表示。为了方便，算法 2 的压缩层函数返回这样的表示。

当从下往上遍历 XTRUCTURE 的层时，我们沿着这个方向传播部分表示。首先，符号层返回单个字符、字符类（例如.、-、：等）或者取决于前一部分中描述的卡方检验的结果的一组字符。然后，附加标记表示的所有符号层表示，从而得到标记，这意味着对于具有层 l_1 至 l_n 的标记表示 k_1，其中在以下中 || 表示串联运算符：

$$\text{str}\,(k_1) = \|_{i=1}^{n}\,\text{str}\,(l_i)$$

然后，这些令牌表示与适当的分隔符（在学习过程期间保持）交织以形成分支表示，假定：str（b_1）= str（k_1）$\|h_1\|$str（k_2）$\|h_2$... 最后，它再次向上传播，整个 XTRUCTURE 的表示就是它所有分支的 OR：

$$R(X) = \text{str}\,(b_1)\,\|\|\|\text{”}\,|\,\|\,\text{str}\,(b_2)\ldots\ \text{”}|\text{”}\|\text{str}\ \ (b_n)$$

4.1.5 度量 XTRUCTURES 之间的相似性

1. 与其他 XTRUCTURES 的比较

XSYSTEM 的比较操作主要依赖于评分拟合和中心极限定理。具体的过程如下：

我们需要不同 XTRUCTURE 表示的结构之间的语法距离函数，例如 D（X1，X2）：R + × R+，返回一对介于 0 和 1 之间的分数，表示每个 XTRUCTURE 的结构与另一个 XTRUCTURE 的” 匹配” 程度。前面我介绍了将元组拟合到 XTRUCTURE 时获得的评分拟合。现在将 XTRUCTURE，X1 到 X2 的拟合定义为拟合 X2 的由 X1 表示的元组集合的平均得分拟合，将 D（S，X2）建模为一个分布，估计具有一定置信度的均值拟合。这将问题从生成所有元组之一减少到生成元组子集之一，该元组子集将允许我们以统计显著的方式估计平均拟合。然而，为了可靠地估计均值拟合，我们需要数据的基本分布，而我们并不知道。我们也不想对这种分布做任何假设：最好是多模态的，最坏是完全不规则的。为了解决这个问题，使用了中心极限定理，按组对分布进行采样近似于 μ（所需平均值）周围的正态分布。因此，为了计算 X1 在 X2 中的拟合，我们从 X1 生成 n 个元组的组，并且计算它们拟合到 X2 中的平均值和标准偏差; 我们使用 95% 置信区间来估计样本量。

2. 与正则表达式比较当比较由 XTRUCTURE X 表示的结构和由正则表达式 R 表示的结构时，我们还希望获得得分拟合的元组：X 和 R 的吻合程度，反之亦然与 XTRUCTURE 之间的比较过程一样，本方法涉及从 XTRUCTURE（或正则表达式）生成元组，然后度量生成的元组与正则表达式（或 XTRUCTURE）的拟合程度。与之前的方法的不同之处在于元组值是二进制的，即，X 要么适合 R 要么不适合 R（反之亦然）。

为了计算 XTRUCTURE 和正则表达式之间的相似性，本方法通过计算 XTRUCTURE 中的结构 X 拟合正则表达式 R 的概率，如下所示：

$$P(fit(X,R)) = \sum_{n=1}^{N} \text{match}(g(X,n),R)/N$$

其中函数 g（X，n）从 X 生成元组，并且如果元组适合 R，则函数 match 返回 1，如果不适合 R，则函数 match 返回 0。抽取的总次数 N 是通过 CLT 的标准应用选择的，这使我们能够将其视为伯努利随机变量 Pfit 的估计值，从而获得一定范围和置信区间内的近似值。

为了计算 R 相对于 X 的适应度，本次使用了现有的库，这些库从现有的正则表达式生成字符串，

通常称为 xeger。使用其中一个类似 xeger 的工具，从正则表达式生成元组，然后在相反的方向应用相同的技术。通过使用这种方法将 XTRUCTURE 与自动标签分配应用程序中已有的正则表达式进行比较。最后获得了良好的结果，之后会在实验结果部分中进行介绍。然而，值得注意的是，相对于 XTRUCTURE-XTRUCTURE 比较方法，该方法有一些局限性。

首先，如果正则表达式太具体，与附近结构的相似性可能会低得违反直觉。例如，精确表示“ABCD”的正则表达式的结构将与表示“ABCE”的 XTRUCTURE 的结构具有低相似性，而如果要比较的两个结构将由 XTRUCTURE 表示，则不是这种情况。其次，由于我们需要从正则表达式生成元组，正则表达式必须是有限的，因此不允许使用通配符。尽管看起来有限制性，但这并不是一个很大的缺点，因为高度结构化的元组往往缺少通配符，这表明缺乏结构。

4.2 实验环境搭建

1. 下载 python 可以去到 python 的官网：https://www.python.org/，建议大家下载 python3 的版本。，据电脑系统选择哪个版本的 python。如果不知道自己的电脑的版本，可以右键我的电脑 > 属性，进行查看。2. 点击运行下载的 exe 文件，按照安装步骤安装即可。3. 安装 Pycharm 开发环境，进入 https://www.jetbrains.com/，点击 developer tools、pycharm、download、community，进行下载安装 pycharm。

4.3 界面分析与使用说明

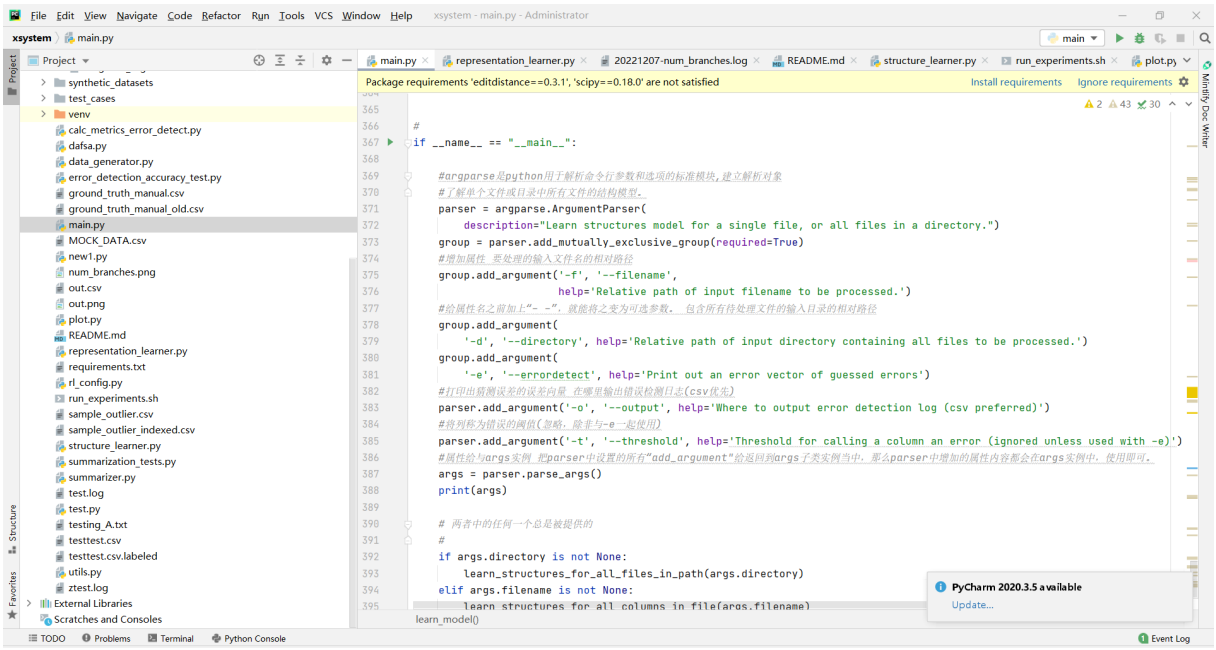


图 2: 操作界面示意图 1

用户可以点击右上角 mian 的下拉按钮，选择 Edit Configurations 打开 run Configurations 窗口，设置需要进行操作的相关参数，然后点击 main 后边的 run 按钮即可。

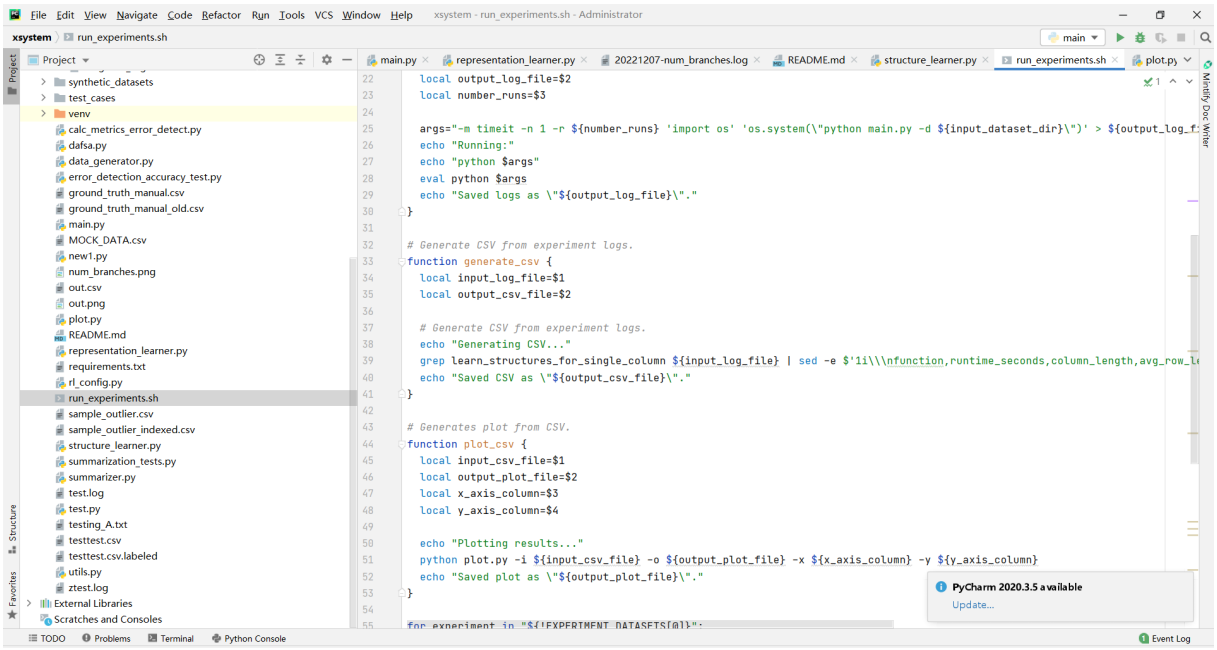


图 3: 操作界面示意图 2

因为操作的步骤和实验数据比较多，所以我在此写了个脚本方便用户更快的使用，用户可以打开 run_experiments.sh 文件，在文件中右击点击 run，然后选择 Git for Windows 即可运行。然后就会出现如下图所示的运行界面，等待运行完毕即可。

```
Running:
python -m timeit -n 1 -r 10 'import os' 'os.system("python main.py -d synthetic_
datasets/fixed_col_size/different_hinges_and_free_form_text-single_column")' > l
ogs/20221207-num_branches.log 2>&1
Saved logs as "logs/20221207-num_branches.log".
Generating CSV...
Saved CSV as "logs/20221207-num_branches.csv".
Plotting results...
   runtime_seconds  num_branches
0           11.900            10
1           16.690            10
2           28.761            10
3           25.630            10
4           28.561            10
    runtime_seconds  num_branches
0            11.900            10
1            16.690            10
2            28.761            10
3            25.630            10
4            28.561            10
12            8.910             9
18           20.400             9
27           35.130             9

quantile_df for quantile 0.99
                runtime_seconds
num_branches
9                     34.83540
10                    53.28926
```

图 4: 操作界面示意图 3

4.4 创新点

1. 采用从示例中逐步学习语法的方法，利用已知实体中存在的分隔符，将提取模式的问题分解为学习由这些分隔符分隔的每个标记的语法。2. 使用了等价于确定性非循环有限状态自动机（DAFSA）的分支线性分布序列的数据结构，用于学习每个令牌的底层数据结构是 DAFSA，DAFSA 比通常用于正则表达式学习的最小确定性有限自动机（DFA）渐进地更容易学习。3. 优化 1：并行学习。在拟合模型时，使用了多个工作器，每个工作器读取不相交的元组集并独立地拟合它们。这样做的好处是利

用了现代体系结构中现成的并行性，但导致每个属性有多个表示。在这一点上，我们可以使用分支合并算法来合并不同构建模型的分支，从而得到一个等价于单个工作者所学习的表示。优化 2：提前停止。当从结构化数据中学习时，大部分计算时间通常用于拟合对最终 XTRUCTURE 没有贡献的元组。经过几个元组之后，我们建模的表示将反映模式，并且不会随着处理更多元组而改变。当模型已经收敛并且在消耗了一定数量的新元组之后不再改变时，我们可以停止学习。为此，我们跟踪在拟合过程中新元组的拟合变化了多少。最初，预计得分会有很大变化，随着时间的推移，得分会下降并变得稳定–特别是当数据是规则的时候。这种技术允许我们跳过大量数据，同时仍然找到良好的近似表示。

5 实验结果分析

1. 实验数据集和设置（1）大学数据仓库（DHW），其由 161 个表和 1690 个属性组成，具有关于大学内的系的信息;

（2）ChEMBL（CHE），一个公共化学品数据库，有 70 个表格和 461 个属性;

（3）data.gov（GOV），美国公开政府数据，由 2250 个 CSV 文件组成；

（4）对于离群值检测实验使用了 KDDCUP 99 和森林覆盖数据集，这是离群值检测中使用的标准数据集。

默认设置：最大分支设置为 3，分支阈值设置为 0.1，捕获阈值设置为 85%。对于所有的实验使用一台 1.8GHz Intel Core i5 和 8 GB RAM 的计算机。

2. 基于语法的异常值检测

本实验是评估 XSYSTEM 检测数据集中单个属性中异常值的能力。使用了 KDDCUP 1999 入侵检测数据集和森林覆盖数据集。对于定量分析，使用了 KDDCUP 中的三种异常值类型，以及森林覆盖数据集。

论文中的实验结果如图 5所示：

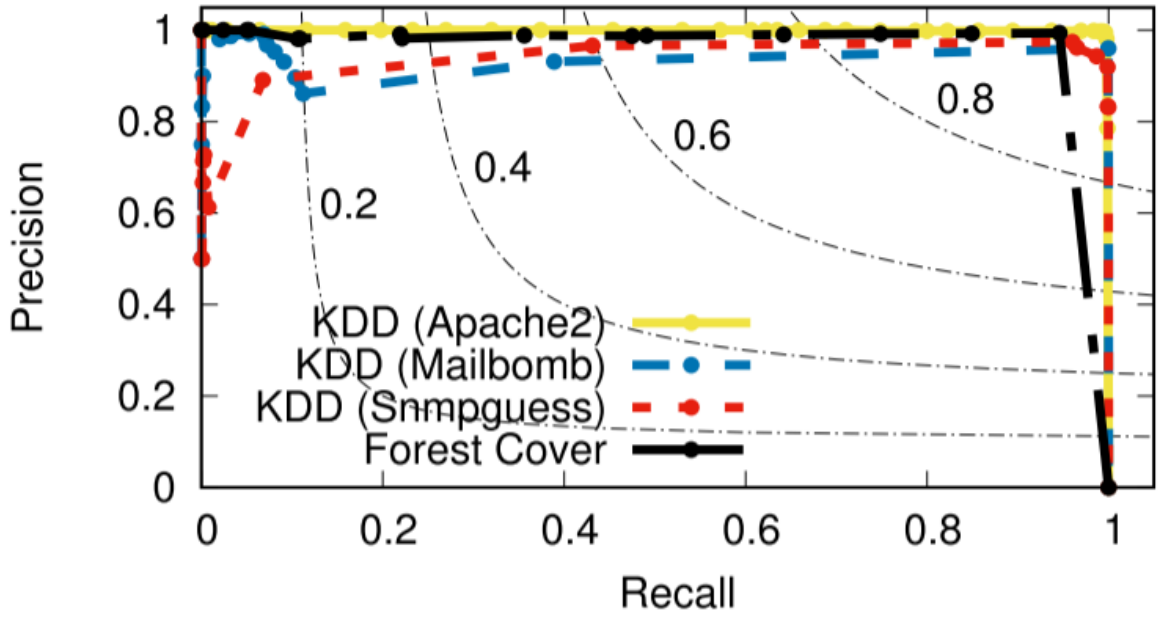
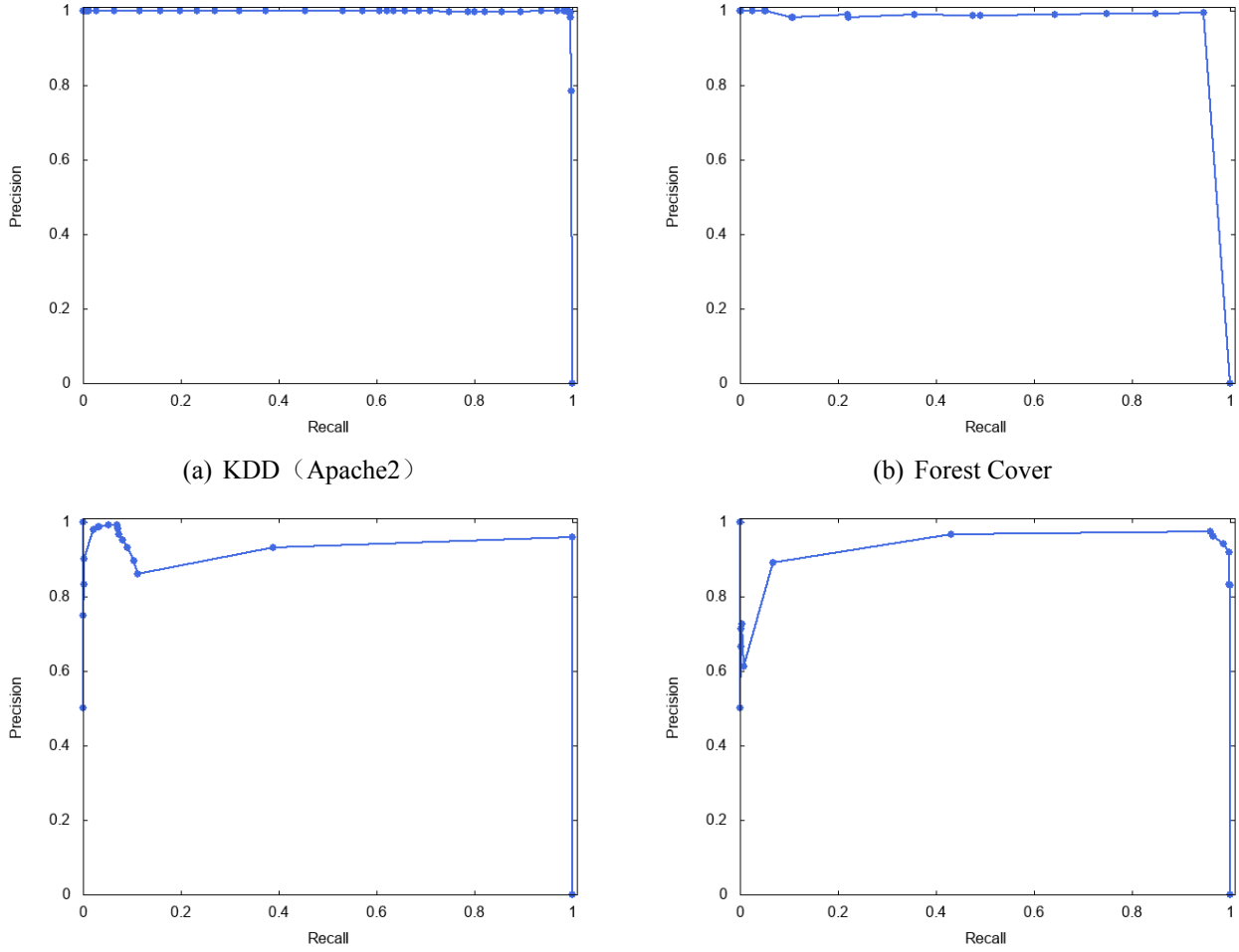


图 5: 论文中的离群值实验的 PR 曲线

本次实验从元组的子集（存在异常值）中学习每个属性的 XTRUCTURE。然后冻结 XTRUCTURE

并向它提供新的元组，获得一个适合度得分，用于查找异常值。使用每个分支评分拟合的加权组合将评分转换为离群值评分。然后标记了超过离群值阈值的离群值分数。本次实验结果如图 6所示。



(a) KDD（Apache2）　(b) Forest Cover

(c) KDD（Mailbomb）　(d) KDD（Snmoguess）

图 6: 离群值实验的 PR 曲线

该图显示了不同离群值阈值的精确度和召回率。结果显示，在所有四个大型数据集上都有近乎完美的性能。图 6（a）是采用 KDD（Apache2）的异常值进行分析而得到的 PR 曲线，该结果显示了采用 KDD（Apache2）的异常值实验得到了极高的的精确度和召回率结果，几无论召回率多少，精确度都是接近于 1。图 6（b）采用的是森林覆盖数据集来进行异常值使用，可以看出曲线形状跟图 6（a）看起来相似，但是结果还是（a）的精确度和召回率较好。高召回率下曲线的轻微不规则性（缺乏平滑度）是由于 LSH 方法基于聚类的性质，而不是直接比较每对属性。这是有意义的，因为我们必须预先生成字符串来生成 MinHash 签名，我们要确保字符串统一地表示底层数据，从而提高签名质量。图 6（c）和图 6（d）采用了 KDD 的另外两种异常值类型来进行实验，KDD（Mailbomb）的类似了。总而言之在召回率 0-0.1 的阶段直接是处于较好的趋势，但之后的精确度就没有之前好了。而 KDD（Snmoguess）在召回率 0-0.1 的阶段是属于落差比较大，精确度比较低，但是随后的数据差不多跟 KDD（Mailbomb）的类似了。总而言之，在所有四个大型数据集上的异常值检测实验都是表明复现的方法有着很好的性能。

3. 微基准实验

元组数、每个元组的分隔符数和模式异构性等属性都会影响 XSYSTEM 的性能。为了测量这些效果，本次实验生成了具有不同属性的合成数据，然后对使用 XSYSTEM 运行这些数据。

论文中的实验结果如图 7所示：

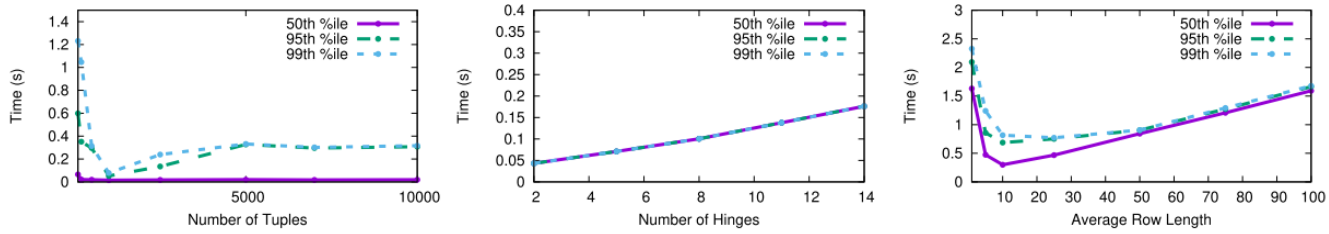
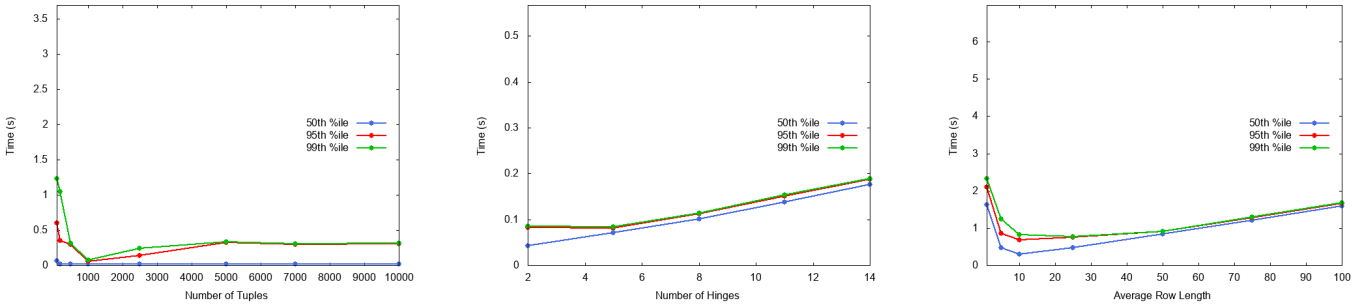


图 7: 论文中的使用合成数据集的性能微基准测试结果

本次实验结果如图 8所示，显示了平均 10 次的运行时间，以及中位数、第 95 和第 99 百分位数。



(a) Number of Tuples (b) Number of Hinges (c) Average Row Length

图 8: 使用合成数据集的性能微基准测试结果

在第一个实验（a）中，使用了固定长度的数据类型（国家货币代码），并改变了输入数据集中元组的数量。在第二个实验（b）中，测量了铰链数量的影响，铰链数量对 XSYSTEM 必须维护的令牌数量有影响。在这里，实验中通过连接 YYYY-MM-DD 格式的日期来创建具有可变数量铰链的输入数据集，并将元组的数量固定为 1000。最后，在第三个实验（c）中，使用了具有 1000 个元组和混合数据类型的数据集来改变属性值长度，这会影响 XSYSTEM 维护的每个 XTRUCTURE 的分支总数。XSYSTEM 在所有 3 个实验中的性能随感兴趣的变量线性增长。尽管第三次实验中的绝对值较高，但第 99 百分位数低于 10 秒，中位数低于 1 秒。总体来说，XSYSTEM 的性能还是相当不错的。

6 总结与展望

本文开始大概介绍了复现论文的相关背景及主要内容，接着讨论了与 Xsystem 相关的若干技术和研究领域的贡献，随后详细介绍了本文的实现方法、复现细节、优化和创新点以及实验环境的搭建，最后通过实验结果的分析来产生本文的结论。在实验过程中，最开始因为编程语言掌握的不太熟练，所以在编写代码中老是会出错，所幸经过一段时间的努力后，代码基本都完成了，但是在跑数据集的时候，最开始找了很久都没找到文中的数据集，自己找了一部分小数据集，结果发现跑出来的结果和文中实验的结果差距太大。后面经过一阵子努力寻找后，终于找到了原始数据集，最后跑出来的结果终于和文中结果类似甚至还稍微好点。XSYSTEM 是一种在数据库列上学习模式的有效方法，所用的时间要少得多，该方法可以在更短的时间内学习数据库列上的模式，使这些模式不仅可以快速构建，而且可以捕获许多关键应用程序，包括检测异常值，测量列相似性以及为列分配语义标签。但是用 XSYSTEM 来学习 XTRUCTURE 数据的语法结构，XTRUCTURE 的表达能力不如正则表达式，但学习速度要快很多，而且表达能力足以表示数据库中常见的高度结构化的数据。我们现在只做句法分析，但是未来可能想加入一些语义实体，比如人名、国名等等, 通过机器学习等方法来进行命名实体识别。

参考文献

[1] BARTOLI A, LORENZO A D, MEDVET E, et al. Inference of Regular Expressions for Text Extraction from Examples[J]. IEEE Transactions on Knowledge & Data Engineering, 2016, 28(5): 1944.

[2] BRAUER F, RIEGER R, MOCAN A, et al. Enabling information extraction by inference of regular expressions from sample entities[C]//Acm Conference on Information & Knowledge Management. 2011.

[3] LI Y, KRISHNAMURTHY R, RAGHAVAN S, et al. Regular Expression Learning for Information Extraction[C]//2008 Conference on Empirical Methods in Natural Language Processing, EMNLP 2008, Proceedings of the Conference, 25-27 October 2008, Honolulu, Hawaii, USA, A meeting of SIGDAT, a Special Interest Group of the ACL. 2008.

[4] FERNAU H. Algorithms for learning regular expressions. (Extended abstract)[J]. Springer, Berlin, Heidelberg, 2005.

[5] BEX G J, NEVEN F, SCHWENTICK T, et al. Inference of Concise Regular Expressions and DTDs[J]. Acm Transactions on Database Systems, 2010, 35(2): 1-47.

[6] Hakjoo, Lee, Mina, et al. Synthesizing Regular Expressions from Examples for Introductory Automata Assignments[J]. ACM SIGPLAN Notices: A Monthly Publication of the Special Interest Group on Programming Languages, 2017, 52(3): 70-80.

[7] LE V, GULWANI S. FlashExtract: A Framework for Data Extraction by Examples[J]. Acm Sigplan Notices, 2014, 49(6): 542-553.

[8] SINGH R. BlinkFill: Semi-supervised Programming By Example for Syntactic String Transformations [J]. Proceedings of the Vldb Endowment, 2016, 9(10): 816-827.