连接上的子集采样

ARYAN ESMAILPOUR,伊利诺伊大学芝加哥分校，美国XIAO HU,滑铁卢大学，加拿大JINCHAO HUANG, 香港中文大学，香港
STAVROS SINTOS,伊利诺伊大学芝加哥分校，美国

arXiv:2512.16321v1 [cs.DB] 18 Dec 2025

子集采样（也称为泊松采样）是一种基本操作：是否将某个特定元素纳入样本的决定独立于其他所有元素。它是数据分析中的基础方法，通过处理具有代表性的子集而非海量数据集，实现高效近似。虽然从显式列表中抽样已广为人知，但现代应用——例如针对关系数据进行机器学习——往往需要从由关系连接隐式定义的集合中抽样。本文研究连接上的子集采样问题：从连接结果中抽取一个随机子集，其中每个连接结果都以某种概率独立纳入。我们研究一般情形，其中概率由输入元组权重通过可分解函数（例如乘积、求和、最小值、最大值）计算得出。由于连接大小可能远远超过输入大小，朴素地将所有连接结果物化后再进行子集采样，在计算上不可行。我们提出了首批针对无环连接的高效子集采样算法：(1) 一种静态索引，用于生成连接上的多个（相互独立的）子集样本；(2) 一种一次性 算法，用于生成连接上的单个子集样本；(3) 一种动态索引，能够支持元组插入，同时维护一次性样本或生成多个（相互独立的）样本。相对于输入大小和预期样本大小，我们的方法实现了近乎最优的时间和空间复杂度。

1 引言

抽样是现代数据分析和机器学习中的基石技术。通过处理具有代表性的数据子集而非整个数据集，系统可以牺牲少量准确性来显著提升性能 [20, 39]。一种尤为强大的基本方法是子集采样 （也称为泊松采样）：根据特定的概率或权重，独立决定是否将任意特定元素纳入样本，而不受其他元素影响。这种独立性对于随机化算法的理论保证通常至关重要，也使分布式系统能够在无需协调的情况下执行 [24, 37]。

从明确列出的项目中进行子集采样已得到充分研究，但在现代数据管道中出现了新的挑战：感兴趣的数据集并非存储在单个表中，而是隐式 定义为关系连接的结果。本文研究连接上的子集采样问题：如何从连接结果中高效抽取随机子集，其中每个连接结果都关联一个权重，并以等于该权重的概率独立纳入样本。至关重要的是，我们关注的是标准情形：连接结果的权重并非任意指定，而是通过聚合函数从输入元组的权重导出，例如对各分量权重取乘积、最小值、最大值或求和。该问题的一个重要动机是对多关系数据进行机器学习时的数据集凝缩（或核心集构造） [33, 41]。

示例1.1。 考虑一项高风险的科学计算任务——药物发现中的虚拟筛选——研究人员希望训练一个图神经网络，以预测复杂分子相互作用的疗效。训练数据通过连接包含化合物、蛋白质靶点和代谢通路的海量表来定义。所有有效相互作用子图的完整集合（连接结果）在组合数量上呈爆炸式增长——规模通常超过PB级——因此不可能

作者联系信息：AryanEsmailpour，伊利诺伊大学芝加哥分校，美国芝加哥，aesmai2@uic.edu；Xiao Hu，滑铁卢大学，加拿大滑铁卢，xiaohu@uwaterloo.ca；Jinchao Huang，香港中文大学，中国香港，jchuang@link.cuhk.edu.hk；Stavros Sintos，伊利诺伊大学芝加哥分校，美国芝加哥，stavros@uic.edu。

将其物化或进行穷举式模拟。为使训练切实可行，系统必须生成一个精简且具有代表性的数据集 [6, 26]。在此过程中，输入元组携带关键的领域特定权重：化合物具有结合亲和力评分，蛋白质靶点具有临床相关性评分。所得相互作用元组的重要性自然由这些组成部分推导而来（例如，亲和力与相关性的乘积，表示成功的联合概率；或最小值，表示瓶颈约束）。通过使用这些派生权重进行子集采样，我们可以创建化学空间的可管理、高质量摘要。

然而，由于数据具有隐式性质，要高效地执行这种抽样在算法层面颇具难度。一种简单的方法是先计算连接结果及其聚合权重（例如，其各组成部分的乘积），然后再应用经典子集采样算法。这种策略代价高得难以承受，因为存在“物化瓶颈”：连接结果的规模可能远大于输入关系 [5], ，而计算和存储这些结果的代价十分高昂。在这种朴素方法中，算法的大部分时间都会耗费在生成连接结果上，而这些结果随即便会被抛硬币的结果丢弃，从而浪费大量计算和内存资源。据我们所知，除了这一基线之外，此前没有研究解决过关系数据的子集采样问题。本文旨在填补这一空白，提出首批可直接在输入关系上运行的高效子集采样算法。通过利用查询的连接结构以及权重函数的可分解性质，我们可以有效地“跳过”大量被拒绝的元组，直接生成成功样本。我们的研究成果共同确立了连接上的子集采样是一种可处理的数据库基本操作，并将为海量多关系数据集上的可扩展交互式分析铺平道路。

1.1 问题定义

子集采样。给定一个元素集合 $S = \{e_1, e_2, \ldots, e_n\}$，以及一个为每个元素 $e \in S$ 分配概率 $\mathrm{p}(e)$ 的函数 p，子集采样查询要求返回 S 的子集样本。 S 的子集样本是一个随机子集 $\mathbf{X} \in 2^S$，其中每个元素 $e \in S$ 都以概率 $\mathrm{p}(e)$ 独立地被采样到 X 中。更正式地说，X 的分布为：$\Pr[\mathbf{X} = Y] = (\prod_{e \in Y} \mathrm{p}(e)) \left(\prod_{e \in S \setminus Y} (1 - \mathrm{p}(e))\right), Y \subseteq S$. 我们将 $\Psi = S, \mathrm{p}$ 称为一个子集采样问题实例。我们将 X 的期望大小，即 $E[|\mathbf{X}|] = \sum_{e \in S} \mathrm{p}(e)$，记为 μ_Ψ。

（自然）连接。设 att 为一个集合，其中每个元素称为属性；dom 为另一个集合，其中每个元素称为值。集合 $U \subseteq \mathrm{att}$ 上的元组是一个函数 $\boldsymbol{u} : U \to \mathrm{dom}$。对于任意子集 $U' \subseteq U$，定义 $\boldsymbol{u}[U']$为集合 U' 上的元组 $\boldsymbol{u}'$，使得对于每个 $A \in U'$，$\boldsymbol{u}'(A) = \boldsymbol{u}(A)$。我们也称 U 为 $\boldsymbol{u}$ 的支撑属性。关系是由同一属性集合 U 上的元组构成的集合 R；我们称 U为 R的模式，这一事实记作 $\mathrm{schema}(R) = U$。给定子集$U \subseteq \mathrm{schema}(R)$，$R$在 U上的投影（记作 $R[U]$）是一个模式为 U 的关系，其定义为$R[U] = \{\text{tuple } \boldsymbol{u} \text{over } U \mid \exists \text{tuple } \boldsymbol{v} \in R \text{s.t. } \boldsymbol{u}[U] = \boldsymbol{v}[U]\}$。我们将自然连接（下文简称“连接”）表示为由关系构成的集合 Q。我们用 $k = |Q|$ 表示 Q 中的关系数量。 Q 的输入大小定义为 Q中的元组总数，记作 $N = \sum_{R \in Q} |R|$。定义 $\mathrm{attset}(Q) = \bigcup_{R \in Q} \mathrm{schema}(R)$。连接结果是 $\mathrm{attset}(Q)$ 上的如下关系：

$$\mathrm{Join}(Q) = \{\text{tuple } \boldsymbol{u} \text{ over } \mathrm{attset}(Q) \mid \forall R \in Q, \boldsymbol{u}[\mathrm{schema}(R)] \in R\}.$$

连接 Q可以用模式图 $G_Q = (V, E)$来描述。它是一种超图，其中 V 中的每个顶点都是 $\mathrm{attset}(Q)$ 中的一个独立属性，而 E 中的每条边都是 Q中一个独立关系的模式。集合 E 可能包含相同的边，因为 Q 中的两个或多个关系可能具有相同的模式。“超”一词表示一条边可以包含两个以上的属性。若 G_Q 是无环的，则连接 Q 是无环的 [44]。如果存在一棵称为连接树的树 T，其节点与 Q 中的关系一一对应（即与 E中的超边一一对应），则该超图是无环的，

对于任意属性 $A \in V$，包含 A 的关系所对应的节点在 T 中构成一个连通子树。非无环的连接称为循环连接。

作为连接的一种变体，R_1 和 R_2 之间的半连接定义为 $R_1 \ltimes R_2 = R_1 \bowtie (R_2[\text{schema}(R_1) \cap \text{schema}(R_2)])$，它会返回 R_1 中所有能与 R_2 中的某个元组连接的元组。当第二个操作数是单个元组 v 时，$R_1 \ltimes v$ 是 $R_1 \ltimes \{v\}$ 的简写。作为一个特例，$R \ltimes \emptyset = R$。

连接上的子集采样。设 $Q = \{R_1, R_2, \ldots, R_k\}$ 是上述定义的连接。每个关系 $R_j \in Q$ 都关联一个函数 p_j：$R_j \to [0,1]$，该函数为其中每个元组分配一个来自 $[0,1]$ 的权重。对于每个连接结果 $\boldsymbol{u} \in \text{Join}(Q)$，其权重是一个定义在构成该连接结果的这些元组权重上的函数 $\mathbf{F}$：

$$\mathrm{p}(\boldsymbol{u}) = \mathcal{F}\left(\mathrm{p}_1(\boldsymbol{u}[\text{schema}(R_1)]), \mathrm{p}_2(\boldsymbol{u}[\text{schema}(R_2)]), \ldots, \mathrm{p}_k(\boldsymbol{u}[\text{schema}(R_k)])\right), \tag{1}$$

其中 F 可以是 MAX、MIN、乘积或求和函数。要求 $\mathrm{p}(\boldsymbol{u}) \in [0,1]$。给定一个连接实例 Q 以及一组关联的权重函数 $\{\mathrm{p}_j\}_{j \in \{1,2,\ldots,k\}}$，关系子集采样查询是针对子集采样实例 $\Psi = \text{Join}(Q), \mathrm{p}$ 的子集采样查询，即要求对 $\text{Join}(Q)$ 中的一个随机子集 $\mathbf{X}$ 进行采样，其中每个连接结果 $\boldsymbol{u}$ 都独立地以其权重 $\mathrm{p}(\boldsymbol{u})$ 所定义的概率被采入 $\mathbf{X}$。更正式地说，$\mathbf{X}$ 的分布为

$$\mathbf{Pr}[\mathbf{X} = Y] = \left(\prod_{\boldsymbol{u} \in Y} \mathrm{p}(\boldsymbol{u})\right)\left(\prod_{\boldsymbol{u} \in \text{Join}(Q) \setminus Y} (1 - \mathrm{p}(\boldsymbol{u}))\right), Y \subseteq \text{Join}(Q). \tag{2}$$

我们将 $\mathbf{X}$ 的期望大小（即 $E[|\mathbf{X}|] = \sum_{\boldsymbol{u} \in \text{Join}(Q)} \mathrm{p}(\boldsymbol{u})$）记为 μ_Ψ。

我们研究的第一个问题是索引（数据结构）问题，即我们希望构建一个索引，以便高效地从连接结果中抽取多个独立的子集样本：

> **问题 1.2**（连接上的索引子集采样）。给定一个连接实例 Q 以及一组关联的权重函数 $\{\mathrm{p}_i\}_{i \in \{1,2,\ldots,k\}}$，目标是构建一个索引，用于回答针对实例 $\text{Join}(Q), \mathrm{p}$ 的子集采样查询。此外，不同查询所返回的随机子集必须相互独立。

我们关注所构建索引的预处理时间 t_p 和空间使用量，以及通过索引回答关系子集采样查询所需的时间 t_s。

我们研究的第二个问题是关系子集采样的一次性版本，我们希望高效地从连接结果中输出一个子集样本：

> **问题 1.3**（连接上的单次子集采样）。给定一个连接实例 Q 以及一组相关的权重函数 $\{\mathrm{p}_i\}_{i \in \{1,2,\ldots,k\}}$，目标是从该实例 $\text{Join}(Q), \mathrm{p}$ 中计算一个子集样本。

在这种情况下，我们关注总回答时间。尽管问题 1.2 的任何解法都能以总时间 $O(t_p + t_s)$ 给出问题 1.3 的解法，但这可能还不够高效。

连接上的动态子集采样。我们还考虑包含元组插入[1]的动态设置。现在，每次插入都是一个三元组 $\boldsymbol{u}$，R_h，η，p，其中 $\eta \in \mathbb{Z}^+$、$p \in [0,1]$ 和 $R_h \in Q$ 分别表示元组 $\boldsymbol{u}$ 在时间戳 η 以权重 p 插入关系 R_h。我们遵循集合语义，因此向一个已包含某元组的关系中插入该元组不会产生任何影响。考虑按时间戳排序的输入元组流。令 Q^η 为由该流前 η 个元组定义的连接，并设 $Q^0 = \emptyset$。此外，令 $\{\mathrm{p}_i^\eta\}_{i \in \{1,2,\ldots,k\}}$ 为与 Q^η 相关的权重函数。每个连接结果 $\boldsymbol{u} \in Q^\eta$ 的概率 $\mathrm{p}^\eta(\boldsymbol{u})$ 按照与（1）类似的方式定义。我们用 N 表示数据流的长度，

[1]在同时允许插入和删除的完全动态情形下，权重全为 1 的子集采样问题至少与维护连接结果一样困难。对于一般的（更准确地说，非层次化的）连接查询，仅为了维护布尔答案，更新时间就必须达到 [7]$(\sqrt{N})$，其中允许同时进行插入和删除Ω。即使对于这个布尔问题，要证明与之匹配的上界仍是一个悬而未决的问题，某些特定连接除外。

仅用于分析。算法不需要知道 N，因此能够处理无界数据流。我们有这两个问题在动态设置下的对应版本：

问题 1.4（连接上的动态索引子集采样）。假设元组以流式方式到达。目标是维护一个索引，以便对每个时间戳 $\eta \in \mathbb{Z}^+$，都能针对实例 $\text{Join}(Q^\eta)$,p^η 回答子集采样查询。此外，不同查询返回的随机子集必须彼此独立。

问题 1.5（连接上的动态一次性子集采样）。假设元组以流式方式到达。目标是针对每个时间戳 $\eta \in \mathbb{Z}^+$，从实例 $\text{Join}(Q^\eta)$,p^η 维护一个子集样本。

对于动态索引，我们关注插入元组时索引的维护时间 t_u，以及通过索引回答关系子集采样查询所需的时间 t_s。对于动态一次性问题，我们关注总运行时间。同样，问题 1.4 的任何解法都能以总时间 $O(t_u \cdot N + t_s)$ 给出问题 1.5 的解法。

对于关系子集采样问题的所有版本，我们研究其数据复杂度 [1]，并根据依赖数据的量（例如输入大小 N和子集样本的期望大小 μ_Ψ）分析复杂度，同时将 Q 的模式规模（即 $|V|$ 和 $|E|$）视为常数。

计算模型。我们在实数 RAM 计算模型 [9, 40]下讨论我们的算法。具体而言，我们假设以下操作均可在常数时间内完成：(i) 访问一个内存位置；(ii) 从标准均匀分布 Uniform(0, 1) 中生成一个随机值；以及 (iii) 执行涉及实数的基本算术运算，例如加法、乘法、除法、比较、截断，以及计算 log 等基本函数。我们用 Geometric(p) 表示定义在$\{0, 1, \ldots\}$上的几何分布，其概率质量函数为 $\Pr[X = k] = (1-p)^k p$。我们用 TruncatedGeometric(p, n) 表示给定其取值严格小于 n条件下的几何分布，即其支持集为$\{0, 1, \ldots, n-1\}$。在 RAM 模型下，我们可以通过令 $x = \lfloor \log \text{Uniform(0,1)} \log(1-p) \rfloor$，在 [10] 时间内生成一个来自 Geometric(p) 的随机值 x，所需时间为 $O(1)$。此外，我们可以通过令 $x = \lfloor \log(1-q \cdot \text{Uniform(0,1)}) \log(1-p) \rfloor$，在 [10] 时间内~~生成一个来自~~ TruncatedGeometric(p, n) 的随机值 x，所需时间为 $O(1)$，其中 $q = 1-(1-p)^n$。

数学约定。对于整数 $x \in \mathbb{Z}^+$，记号 $[x]$ 表示集合 {1，2，……，x}，而 $[\![x]\!]$表示集合 {0，1，……，x}。给定元素的有序集合 X以及一对元素 x，$x' \in X$，我们用 $x \prec x'$ 表示 x 小于 x'，用 $x \preceq x'$ 表示 x 不大于 x'。我们用 $\log x$ 表示 $\log_2 x$。

为了便于理解，我们在表 2 中列出了本文使用的主要记号。

1.2 相关工作

连接上的均匀采样。Chaudhuri 等人 [13] 和 Olken [39] 首次提出了连接上的均匀采样问题，该问题旨在从连接结果中返回一个均匀分布的样本。Zhao 等人 [46]提出了一种线性空间索引，用于从无环连接中抽取均匀分布的独立样本。随后，Chen 和 Yi [15] 研究了循环连接，其结果经两项独立研究 [23, 32]改进，达到了条件最优。最近，Dai 等人 [22] 提出了一种蓄水池采样框架，用于在流式场景中维护连接上的均匀分布样本。

连接上的非均匀采样。在线聚合（OLA）领域已对非均匀采样进行了广泛研究。涟漪连接 [28]及其变体将嵌套循环连接推广为增量式估计聚合量，并给出置信区间。同样，漫游连接 [34]通过在连接图上进行随机游走，为求和或计数等聚合量提供无偏估计量。这些方法与我们的做法有本质区别：它们旨在使用能够最小化方差的算法概率来估计标量

静态	方法	预处理时间	采样时间	空间使用量
索引	基线	$O(N + \|连接(Q)\|)$	$O(1 + \mu_\Psi)$	$O(\|连接(Q)\|)$
	我们的算法	$O(N$对数 N对数 对数 $N)$	$O(1 + \mu_\Psi$ 对数 $N)$	$O(N$对数 $N)$
一次性	基线	–	$O(N + \|连接(Q)\|)$	$O(\|连接(Q)\|)$
	我们的算法	–	$O(N$ 对数^2 $N + \mu_\Psi)$	$O(N$ 对数^2 $N + \mu_\Psi)$
动态	**方法**	**更新时间**	**采样时间**	**空间使用量**
索引	我们的算法	$O($对数^3 N对数对数 $N)$	$O(\mu_\Psi$ 对数 $N)$	$O(N$对数 $N)$
一次性	我们的算法	–	$O(N\log^3 N \log\log N + \mu_\Psi \log N)$	O（N对数 N）

表1. 乘积函数下无环连接的基线算法和我们的算法的复杂度。N 是输入大小，$|\text{join}(Q)|$ 是连接大小，μ_Ψ 是期望输出大小。

统计量，而我们的目标是根据用户定义的重要性权重，物化一个由独立样本构成的具体子集。

尽管关于连接上的抽样已有大量研究，但据我们所知，目前尚无任何现有方法可以调整为支持连接上的子集采样，这表明需要新的技术。一种直接的连接上子集采样方法是先物化连接结果，然后为所有输出元组建立标准索引。然而，连接结果的大小可能比输入数据库大几个数量级，因此这种方法的效率低得难以接受。更广泛地说，越来越多的研究致力于在不完全物化连接结果的情况下，求解连接结果上的优化问题 [2–4, 12, 14, 16, 17, 21, 25, 30, 31, 33, 35, 36, 41–43]。

1.3 我们的结果

我们的主要结果总结见表1。本文聚焦于无环连接，我们的所有算法都可以通过标准的树分解方法扩展到循环连接，但代价是输入大小增加。我们的贡献针对三种主要场景：

- **（第 3 节）**对于静态场景，我们设计了一种索引，可在 O（$N\log N \log\log N$）时间内构建，占用 O（$N\log N$）空间，并支持在 O（$1 + \mu_\Psi \log N$）期望时间内执行子集采样查询，其中 N 为输入大小，μ_Ψ 为期望样本大小。
- **（第 4 节）**接着，我们提出一种单次算法，无需构建索引，即可在 O（$N\log^2 N + \mu_\Psi$）期望时间内生成一个子集样本。
- **（第 5 节）**最后，我们通过应用并改造 [22]中的无环连接动态直接访问索引，将我们的框架扩展到支持插入的动态设置。

备注 1. 在正文中，我们假设函数 F 是输入权重的乘积。在附录E中，我们将展示如何通过简单地调整我们的算法来支持其他函数。

1.4 作为预备知识的先前结果

子集采样。 针对给定元素集合的经典子集采样问题已得到充分研究 [8, 10, 29, 45]。一种朴素的方法是遍历所有 n 元素，并为每个元素抛一次偏置硬币，耗时 $O(n)$。因此，目标是设计查询时间与期望输出大小 μ_Ψ成正比的算法，而后者可能远小于元素总数。在动态设置中，元素和概率可能发生变化，任何算法都需要 $\Omega(1+\mu_\Psi)$ 查询时间和 $\Omega(1)$ 更新时间 [45]。在静态场景中，[10] 提出了首个具有最优查询时间 $O(1 + \mu_\Psi)$ 的索引。然而，他们的索引需要 $O(\log^2 n)$ 更新时间，在动态设置中并非最优。最近，[8, 29, 45] 通过索引以最优方式解决了这一动态问题，其期望查询时间为 $O(1 + \mu_\Psi)$，空间为 $O(n)$，更新时间为 $O(1)$。

[2]输入大小 其中 N将替换为 N^{fhtw}，fhtw 是模式图 G_Q的分数超树宽度 [27] .

最坏情况最优连接。 AGM 界指出，对于大小为 N的任意连接实例 Q，连接结果的最大数量为 $\Theta(N^{\rho^*})$ [5]，，其中 ρ^* 是模式图 G_Q的分数边覆盖数。最坏情况最优连接算法可以在 $O(N^{\rho^*})$ 时间 [38]内计算大小为 N的任意连接实例 Q。

连接的直接访问。 连接的直接访问 问题最早由 Carmeli 等人研究，他们假设 Join(Q) 的连接结果有一个固定顺序 [11]，，并要求建立一个索引，以返回某个特定位置上的连接结果。对于任意无环连接 Q，可以在 $O(N)$ 时间内建立索引，使得每个直接访问查询都能在 $O(\log N)$ 时间内得到回答 [11, 46]。

2 重新审视经典子集采样

本节重新审视经典子集采样问题，这是我们关系算法的基础。我们依赖于针对输入集合 S（在某个固定顺序[3]下）的 DirectAccess 预言机这一概念。给定一个整数 $i \in [|S|]$，该预言机会在 O 中以 S 时间返回第 i 个元素。所有缺失的证明均见附录 B。

2.1 完美均匀子集采样

子集采样问题实例 $\Psi = S$，p 若满足：对于任意 $e \in S$，$\mathrm{p}(e) = p$，则称其为完全均匀 。这也可简写为 $\Psi = S$，p。对于完全均匀的情形，我们有以下经典结果：

引理 2.1 (完全均匀子集采样)。 对于完全均匀的子集采样实例$\Psi = \langle S, p\rangle$，可以构建索引 $\mathcal{D}$，其所需空间和时间为 $O(|S|)$；利用该索引，每个子集采样查询的期望时间为 $O(1 + \mu_\Psi)$。

索引 D 由一个数组构成，用于存储 S 的元素，这使我们能够在 $O(1)$ 时间内检索第 i 个元素。此外，我们预先计算结果样本非空的概率 $q = 1 - (1-p)^{|S|}$。构建该索引需要 $O(|S|)$ 的时间和空间。借助此索引，有两种算法可以达到相应的查询时间复杂度。算法 1 采用标准方法，通过几何跳跃越过被拒绝的元素。然而，算法 2 利用预先计算的概率 q 引入一个前置步骤：先掷一枚硬币，决定是否从S中至少抽取一个元素。对于单个实例而言，两种算法都是最优的；但在处理多个完美均匀实例时，算法 2 具有一项特殊优势。我们无需对每个实例都调用采样过程，而是可以先抽取能产生非空结果的实例子集，然后仅在这些实例中进行子集采样。在批处理场景中，这种策略能显著降低总时间复杂度，详见第 2.3 节。

算法 1：uss-vanilla$(S,\ p)$

```
1 X ← ∅; i ← 0;2 当 i < |S| 执行
3     i ← i + 1 +几何分布(p);4 将直接访问 (S, i) 添加到 X;
5 返回 X;
```

算法 2：uss-advanced(S, p)

```
1 X ← ∅; q ← 1 − (1 − p)^|S|;
2 若均匀分布(0,1) ≤ q 则
3     i ← 1 + 截断几何分布(p, |S|);
4     将直接访问 (S, i) 添加到 X;
5     当 i < |S| 时
6         i ← i + 1 +几何分布(p);
7         将直接访问 (S, i) 添加到 X;
8 返回 X;
```

[3]排序可以任意指定，但在多次调用此预言机时必须保持一致。

算法 3：ss-rejected-batch（Ψ_i）$_{i\in[m]}$）

输入： 隐式访问 q 的元索引和 $\{\Psi_i\}$ 的子索引。

输出： 一个子集样本X。

$\mathbf{I} \leftarrow$**ss-query**(Ψ_{meta})；// 查询元索引

对 $i \in \mathbf{I}$ **执行**

  $\mathbf{X}_i \leftarrow \emptyset$;

  $j \leftarrow 1 +$截断几何分布$(p_i^+, |S_i|)$；// 使用子索引模拟算法 2

  将直接访问 (S_i, j) 添加到 $\mathbf{X}_i$;

  当 $j < |S_i|$ **时执行**

    $j \leftarrow j + 1 +$几何分布(p_i^+);

    如果 $j \le |S_i|$**那么** 将直接访问 $(S_i,\ j)$ 添加到 $\mathbf{X}_i$;

  对于每个 $e \in \mathbf{X}_i$ **执行** 从 e 中移除 $\mathbf{X}_i$，概率为 $1 - \mathrm{p}_i(e)\ \frac{}{p_i^+}$;

返回 $\bigcup_{i\in\mathbf{I}} \mathbf{X}_i$;

2.2 基于拒绝的子集采样

完美均匀条件对于一般应用而言过于严格。我们可以采用基于拒绝的策略放宽这一要求，而不影响渐近运行时间。给定一个一般实例 $\Psi = S,\ \mathrm{p}$，我们定义一个上界 $p_{\max} = \max_{e\in S} \mathrm{p}(e)$。我们为作为界定对象的完美均匀实例 $\Psi^+ = S,\ p_{\max}$构建引理 2.1 中描述的索引。此预处理需要 $O(|S|)$ 时间。

查询过程首先使用索引（通过算法 1 或算法 2）从 Ψ^+ 中获取一个子集样本 X。然后，对于 X 中的每个元素 $e' \in$，以概率 $\mathrm{p}(e')/p_{\max}$ 将其保留在最终样本中。总运行时间与中间样本 X 的大小成正比。在两种条件下，此运行时间可以得到有效界定：

引理 2.2(β-均匀子集采样)。 给定一个子集采样实例 $\Psi = \langle S, \mathrm{p}\rangle$，如果 最大值$_{e\in S}\mathrm{p}(e) \le \beta \cdot$最小值$_{e\in S}\mathrm{p}(e)$ 对于某个参数 $\beta \ge 1$ 成立，则存在一个索引 $\mathcal{D}$，可在 $O(|S|)$ 空间和时间内构建。利用该索引，每个子集采样查询都能在 $O(1 + \beta \cdot \mu_\Psi)$ 期望时间内得到回答。

引理 2.3 (轻量子集采样)。 给定一个子集采样实例 $\Psi = \langle S, \mathrm{p}\rangle$，如果 最大值$_{e\in S}\mathrm{p}(e) \le 1/|S|$，则存在一个索引 $\mathcal{D}$，可在 $O(|S|)$空间和时间内构建。利用该索引，每个子集采样查询都能在 $O(1)$ 期望时间内得到回答。

2.3 批量基于拒绝的子集采样

考虑一个一般的子集采样问题实例 $\Psi = S,\ \mathrm{p}$，它被划分为一组m互不相交的子实例 $\{\Psi_i = S_i,\ \mathrm{p}\}_{i\in[m]}$，其中 $S = \bigsqcup_{i\in[m]} S_i$。每个子实例都有一个关于最大概率的上界 p_i^+（即，最大值 $_{e\in S_i}\mathrm{p}(e) \le p_i^+$），并且属于轻量或 β-均匀分布。

一种朴素的方法是为每个子实例构建第 2.2 节中的索引，并分别调用其查询过程。然而，这会使每次查询耗费 $\Omega(m)$ 时间，因为我们至少必须花费 $O(1)$ 时间检查每个子实例。为了提高效率，我们构建了一个支持两阶段采样过程的复合索引：

- **预处理阶段：** 首先，对于每个子实例 Ψ_i，我们按照引理 2.1 中的描述，为 $S_i,\ p_i^+$ 构建标准索引。其次，我们构建一个元索引，以确定哪些子实例可能对最终样本有所贡献。我们定义概率函数 $\mathrm{q}: [m] \to [0, 1]$，其中 $\mathrm{q}(i) = 1 - (1 - p_i^+)^{|S_i|}$。注意，$\mathrm{q}(i)$ 是从 S_i 均匀

概率 p_i^+抽样得到非空集合的概率。我们为元实例 $\Psi_{\text{meta}} = [m]$，q 构建一个最优子集采样索引（例如 [8]）。总预处理时间与 $|S|$ 成线性关系。

• **查询阶段：** 算法 3 分两个阶段执行。首先，它查询元索引以获取一组随机索引 $\mathbf{I} \subseteq [m]$。然后，仅针对选定的索引 $i \in \mathbf{I}$，它在子实例 Ψ_i 上调用查询过程（使用算法 2 的逻辑），并执行拒绝抽样。

引理 2.4 (批量子集采样索引)。 给定如上定义的分区实例，可以在 $O(|S|)$ 时间和空间内构建一个索引。使用该索引，算法 *3* 可在 $O\left(1 + \sum_{i=1}^{m} |S_i| \cdot p_i^+\right)$ 期望时间内返回一个有效的子集样本。

3 连接上的索引子集采样

我们从连接实例 Q和相关的权重函数 $\{\mathsf{p}_i\}_{i\in[k]}$入手，考虑问题 1.2。我们假设每个关系 $R_i \in Q$ 中的元组都按某种任意但固定的顺序排列，且关系 Q 也按某种任意但固定的顺序排列。本节所有未给出的证明均见附录 C。

3.1 首次尝试

到目前为止，我们首先假设输入连接实例 Q（采用某种固定顺序）有一个可用的直接访问预言机，该预言机接收一个整数 $i \in [|\text{Join}(Q)|]$，并返回 $\text{Join}(Q)$ 的第 i 个元素。与之前类似，顺序可以任意指定，但在多次调用该预言机时必须保持一致。令 $L = \lceil 2\rho^* \log N \rceil$，其中 ρ^* 是模式图 G_Q的分数边覆盖数。注意，$2^L \geq |\text{Join}(Q)|$。

步骤 1：划分输入关系。对于每个关系 $R_i \in Q$，我们根据元组的概率将其划分为 $L+1$ 个子关系 $R_i^{\langle 0\rangle}$，$R_i^{\langle 1\rangle}, \ldots, R_i^{\langle L\rangle}$，其中 $R^{\langle j}{}_i = \{\boldsymbol{u} \in R_i \mid 2^{-j-1} < \mathsf{p}_i(\boldsymbol{u}) \leq 2^{-j}\}$ 对于 $j \in [\![L-1]\!]$，而 $R_i^{\langle L\rangle} = \{\boldsymbol{u} \in R_i | \mathsf{p}_i(\boldsymbol{u}) \leq 2^{-L}\}$。接着，每种组合 $\boldsymbol{j} = (j_1, j_2, \ldots, j_k) \in [\![L]\!]^k$都定义了 Q的一个子连接，记作：$Q_{\boldsymbol{j}} = \{R^{\langle j_1}1, R^{\langle j_2}2, \ldots, R^{\langle j_k}{}_k\}$。$Q_{\boldsymbol{j}}$的连接结果构成 Q的连接结果的一个不相交划分，即 $\text{Join}(Q) = \bigsqcup_{\boldsymbol{j}\in[\![L]\!]^k} \text{Join}(Q_{\boldsymbol{j}})$。为了回答对子集采样实例 $\Psi = \text{Join}(Q), \mathsf{p}$ 的查询，我们可以返回从所有子实例 $\Psi_{\boldsymbol{j}} = \langle \text{Join}(Q_{\boldsymbol{j}}), \mathsf{p}\rangle$（对于 $\boldsymbol{j} \in [\![L]\!]^k$）中抽取的子集的不相交并集。总共有 $(L+1)^k$ 个子实例。

步骤 2：对所有子实例应用子集采样。我们首先指出这种划分中每个子实例的如下性质：

引理 3.1 (轻型或近似均匀子实例)。设 $\boldsymbol{j} = (j_1, j_2, \ldots, j_k)$。如果 $\sum_{i\in[k]} j_i \geq L$，则实例 $\Psi_{\boldsymbol{j}}$ 是轻量的；否则，它是 2^k-均匀分布的。

为了从这些子实例中高效地抽样，我们采用第 2.3 节中的复合索引策略。我们将子连接 $\{Q_{\boldsymbol{j}}\}_{\boldsymbol{j}\in[\![L]\!]^k}$ 视为互不相交的子实例。我们定义元概率 $\mathsf{q}(\boldsymbol{j}) = 1 - (1 - 2^{-\sum_{i\in[k]} j_i})^{|\text{连接}(Q_{\boldsymbol{j}})|}$，并为实例 $[\![L]\!]^k$,q 构建元索引。这样便能高效跳过空的或未选中的子实例。形式化地说：

• **预处理阶段：** 我们根据权重对输入元组进行划分。对于每个子实例 $Q_{\boldsymbol{j}}$，我们构建一个直接访问预言机（子索引），并计算连接大小（例如，[46]）。此外，我们按照第 2.3 节所述为 q 构建元索引。

• **查询阶段：** 我们使用构建好的元索引和子索引调用算法 3。该算法首先使用元索引抽取一组索引$\mathbf{I}$，然后通过所选子实例 $Q_{\boldsymbol{j}}$ 对应的直接访问预言机检索元组。

注意，我们将每个关系划分为 $L+1$ 个子关系，由此定义出 $(L+1)^k$个子实例。我们分别为每个子实例构建 DirectAccess 索引 [46]，因此其开销受这些子实例的总大小所限，即 $O(N(L+1)^{k-1}) = O(N\log^{k-1} N)$。在回答查询时，DirectAccess 预言机可在 $O(\log N)$ 时间内提供支持，并且每个子实例

都是 2^k-均匀分布的，且所有子实例构成原始连接结果的一个划分，因此根据引理 2.2，对所有子实例进行子集采样的开销为 $O(1+\mu_\Psi \log N)$，其中令 $\beta = 2^k$。综合以上结果，我们得到第一个结论：

定理 3.2. 给定一个无环连接实例 Q 以及一组相关的权重函数 $\{\mathrm{p}_j\}^{R_j \in Q}$，存在一个索引 $\mathcal{D}$，可在 $O(N\log^{k-1} N)$ 空间和时间内构建；利用该索引，每个子集采样查询都能在 $O(1+\mu_\Psi \log N)$ 期望时间内得到回答。

接下来，我们在这一初步结果的基础上进一步改进，展示一种优化索引，它在保持相同查询时间的同时，大幅降低空间使用量和预处理时间。

3.2 连接上子集采样的优化索引

第 3.1 节中的框架证明了该方法可行，但空间效率不高，因为它将权重桶 $\boldsymbol{j} \in [\![L]\!]^k$ 的每种组合都视为一个不同的子实例。现在，我们提出一种优化索引，将空间使用量降至 $O(N\log N)$。

按分数合并子实例。 回顾引理 3.1，子实例Q_j（无论是轻量还是近似均匀）的分类仅取决于索引之和 $\sum_{i=1}^k j_i$。这一观察使我们能够合并子实例。我们不再维护 $(L+1)^k$ 个互不相交的分区，而是根据连接结果的总权重“分数”对其进行分组。

对于关系 R_i 中的任意元组 $\boldsymbol{u}$，令其 分数为 $\phi(\boldsymbol{u}) = \lfloor -\log \mathrm{p}_i(\boldsymbol{u}) \rfloor$。对于连接结果 $\boldsymbol{u} \in \mathrm{Join}(Q)$，其 分数为组件分数之和： $\bar{\phi}(\boldsymbol{u}) = \sum_{i=1}^k \phi(\boldsymbol{u}[\mathrm{schema}(R_i)])$。我们根据该分数，将连接结果 $\mathrm{Join}(Q)$ 划分为不相交桶。令 $\mathrm{B}_\ell = \{\boldsymbol{u} \in \mathrm{Join}(Q) \mid \phi(\boldsymbol{u}) = \ell\}$ 表示分数为 ℓ 的连接结果集合。我们将这些桶视为第 2.3 节批处理框架所需的不相交子实例。具体而言：

(1) **桶 $\ell < L$：** 每个桶 B_ℓ 包含概率范围为 $(2^{-\ell-1},\ 2^{-\ell}]$ 的连接结果。因此，B_ℓ 构成一个上界为 $p_\ell^+ = 2^{-\ell}$ 的 2- 均匀子实例。

(2) **尾部桶 $\ell \geq L$：** 我们将所有分数为 $\ell \geq L$ 的连接结果合并到单个尾部桶 $\mathrm{B}_{\geq L} = \bigsqcup_{\ell \geq L} \mathrm{B}_\ell$ 中。由于任何 $\boldsymbol{u} \in \mathrm{B}_{\geq L}$ 的 $\mathrm{p}(\boldsymbol{u}) \leq 2^{-L} \leq 1/|\mathrm{Join}(Q)|$，这个合并后的桶是一个轻量 子实例（引理 2.3），其上界为 $p_{\geq L}^+ = 2^{-L}$。

算法概述。 我们对这些 $L+1$ 子实例应用算法 3。

• **预处理阶段：** 我们计算每个 $\ell < L$ 的大小 $|\mathrm{B}_\ell|$，以构建元索引。对于尾部桶 $\mathrm{B}_{\geq L}$，我们不维护精确计数或索引；在元索引中，我们仅为其分配一个权重代理值 N^{ρ^*}（连接大小的上界），或者通过回退机制处理，因为它的选择概率可以忽略不计。

• **查询阶段：** 我们将 $\{\mathrm{B}_\ell, \mathrm{p} \mid \ell < L\}$ 和尾部桶作为输入调用算法 3。– 如果元采样选中了带有 $\ell < L$ 的桶 B_ℓ，我们会使用专用的 DirectAccess 预言机（下文将介绍）在 $O(\log N)$ 时间内检索 B_ℓ 中给定索引对应的结果。– 如果选中了尾部桶 $\mathrm{B}_{\geq L}$（这种情况发生的概率极低），我们会即时物化 $\mathrm{B}_{\geq L}$ 中所需的结果，以支持检索。

挑战在于如何在不物化桶 $\mathrm{B}_\ell (\ell < L)$ 的情况下，高效支持对它们的直接访问。接下来，我们将介绍如何实现这一点。

带符号说明的连接树。 如前所述，任何无环连接 Q 都有一个连接树 T，使得：(1) Q 中的关系与 T 中的节点一一对应；(2) 对于每个属性x，包含 x 的节点集合构成一个连通子树。略微滥用符号，我们也使用 R_i 表示关系 $R_i \in Q$在 T 中对应的节点。为简单起见，我们假设 T 中的节点按照中序遍历排序。对于节点 R_i，如果 R_i 不是叶节点，则令 C_i 表示 R_i 的子节点。对于节点 $R_j \in \mathrm{C}_i$，令 $R_{\mathrm{next}(j)}$表示 C_i中紧跟在 R_j之后的子节点。

对于最大的子节点 R_{j^*}，我们将 next(j^*) =设为 null。对于节点 R_i，如果 R_i不是根节点，我们用 parent(i) 表示 R_i的（唯一）父节点。令 key(i) =schema(R_i) ∩schema($R_{\text{parent}(i)}$)表示关系 R_i与其父节点之间的（公共）连接属性，前提是 R_i不是根节点。对于根节点 r，key(r) = ∅。对于节点 R_i，令 T_i表示以节点 R_i为根的子树。对于任意子节点 $R_j \in \mathsf{C}_i$，我们将 T_i^j 定义为 T_i 的子树，其中排除了以 C_i 中位于 R_j之前的子节点为根的所有子树，即 $\mathsf{T}_i^j = \mathsf{T}_i \setminus \cup_{R_{j'} \in C_i:\ R_{j'} \prec R_j} \mathsf{T}_{j'}$。为完整起见，我们还定义 $\mathsf{T}_i^{\emptyset} = \mathsf{T}_i$。

数据结构。 首先，我们从数据库实例中移除所有悬空元组。对于每个关系$R_i \in Q$，我们将输入元组存储在哈希表中，因此，对于任意属性子集 $U \subseteq \text{schema}(R_i)$ 和元组 $t \in \textbf{dom}(U)$，我们都能在 $O(1)$ 时间内获取元组 $R_i \ltimes t$ 的列表。我们任取一棵适用于 Q 的连接树 T，并存储以下统计量以扩充 T。

W-值。 考虑任意内部节点 $R_i \in Q$。对于每个元组 $\boldsymbol{u} \in R_i$，我们存储 L计数器 $W_i^j{}_{,\boldsymbol{u}}(\ell)$，其中 $\ell \in [\![L-1]\!]$ 专门针对每个子节点 $R_j \in \mathsf{C}_i$；$W_i^j{}_{,\boldsymbol{u}}(\ell)$ 存储由子树 T_i^j 中的关系产生、且同时包含 $\boldsymbol{u}$ 的连接结果数量，其分数为 ℓ，即：

$$W_{i,\boldsymbol{u}}^j(\ell) = \left| \left\{ \boldsymbol{t} \in (\bowtie_{R_h \in \mathcal{T}_i^j} R_h) \ltimes \boldsymbol{u} : \sum_{R_h \in \mathcal{T}_i^j} \phi(\boldsymbol{t}[\text{schema}(R_h)]) = \ell \right\} \right|. \tag{3}$$

此外，我们定义并存储 $W_i^{\emptyset}{}_{,\boldsymbol{u}}(\ell)$，即由子树 T_i中的关系产生、同时也包含 $\boldsymbol{u}$且分数为 ℓ的连接结果数量，方法是将（3）中的 T_i^j替换为 $\mathsf{T}_i^{\emptyset}$。为完整起见，对于最大的节点 $R_{j^*} \in \mathsf{C}_i$，我们还定义 $W\text{next}(j^*)_{i,\boldsymbol{u}}(0) = 1$和 $W\text{next}(j^*)_{i,\boldsymbol{u}}(\ell) = 0$，对每个 $\ell \in [L]$均如此。此外，对于每个分数 $\ell \in [\![L-1]\!]$和每个子节点 $R_j \in \mathsf{C}_i$，我们按照 R_i中元组的预定顺序，为 $W_i^j{}_{,\boldsymbol{u}}(\ell)$：$\boldsymbol{u} \in R_i$建立一个前缀和数组。对于每个叶节点 $R_i \in Q$，我们仅为每个元组 $\boldsymbol{u} \in R_i$存储计数器 $W_i^{\emptyset}{}_{,\boldsymbol{u}}(\ell)$。

M值。对于非根节点 R_i，我们将 R_i 投影到 key(i) 上，得到 $R_i[\text{key}(i)]$。对于每个元组 $\boldsymbol{v} \in R_i[\text{key}(i)]$,，我们还计算 $M_{i,\boldsymbol{v}}(0)\, M_{i,\boldsymbol{v}}(1)$ ……，$M_{i,\boldsymbol{v}}(L-1)$：对于 $\ell \in [\![L-1]\!]$，$M_{i,\boldsymbol{v}}(\ell)$ 存储所有元组 $\boldsymbol{u} \in R_i \ltimes \boldsymbol{v}$ 上计数器 $W_i^{\emptyset}{}_{,\boldsymbol{u}}(\ell)$ 的总和，即，

$$M_{i,\boldsymbol{v}}(\ell) = \sum_{\boldsymbol{u} \in R_i \ltimes \boldsymbol{v}} W_{i,\boldsymbol{u}}^{\emptyset}(\ell). \tag{4}$$

我们简要说明如何预处理这些数据结构（完整细节见附录 C）。我们以自底向上的方式计算W/M值。一旦W值计算完毕，就可以直接根据式（4）计算 M值。考虑叶节点 R_i，对于任意 $j \in [\![L-1]\!] \setminus \{\phi(\boldsymbol{u})\}$，我们有 $W_i^{\emptyset}{}_{,\boldsymbol{u}}(\phi(\boldsymbol{u})) = 1$和$W_i^{\emptyset}{}_{,\boldsymbol{u}}(j) = 0$。考虑内部节点 R_i。假设每个子节点 $R_j \in \mathsf{C}_i$ 的 W值和M值均已定义。我们按照 C_i 中节点的递减顺序计算 $W_i^j{}_{,\boldsymbol{u}}$，并遵循递归结构。如果 $\ell < \phi(\boldsymbol{u})$，则 $W_i^j{}_{,\boldsymbol{u}}(\ell) = 0$，否则，

$$W_{i,\boldsymbol{u}}^j(\ell) = \sum_{(\ell_1,\ell_2) \in [\![\ell - \phi(\boldsymbol{u})]\!]:\ell_1+\ell_2=\ell-\phi(\boldsymbol{u})} M_{j,\boldsymbol{u}[\text{key}(j)]}(\ell_1) \cdot W_{i,\boldsymbol{u}}^{\text{next}(j)}(\ell_2). \tag{5}$$

注意，$W_i^{\emptyset}{}_{,\boldsymbol{u}}(\ell)$ 可以通过求助于第一个子节点，以类似的方式计算。重要的是，借助快速傅里叶变换 [18, 19],，可以加速 (5) 的计算，整个计算仅需 $O(NL\log L)$ 时间。最后，根节点 r 的每个分数 $\ell \in [\![L-1]\!]$ 对应的桶大小 $|\mathsf{B}_\ell|$ 可以改写为 $|\mathsf{B}^\ell| = \sum_{\boldsymbol{u} \in R_r} W_r^{\emptyset}{}_{,\boldsymbol{u}}(\ell)$。这一阶段的伪代码见算法 6。**用于 直接访问的算法**。我们的目标是给定分数$\ell \in [\![L-1]\!]$和排名 $\tau \in [|\mathsf{B}_\ell|]$，从桶 B_ℓ 中取回第 τ个元组。为确保“第τ个”元组定义明确，我们采用规范的

算法 4：RecursiveAccess(i, j, v, ℓ, τ)

输入： $i, j \in [k]$ 使得 $R_j \in \mathsf{C}_i$，元组 v，分数 $\ell \in [\![L-1]\!]$，整数 $\tau \in \mathbb{Z}^+$。

输出： 连接结果中第 τ 个元组，其来自 $\mathrm{T}_i^{\,j}$，分数为 ℓ，且可与 v 连接。

1 $u \leftarrow R_i \ltimes v$ 中最小的元组，使得 $\sum_{u' \in R_i \ltimes v:\, u' \preceq u} W_i^{\,j}{}_{,u'}(\ell) \geq \tau$；

2 **如果** R_i 是叶节点，**则返回** u；

3 $\tau \leftarrow \tau - \sum_{u' \in R_i \ltimes v:\ u' \prec u} W_{i,u'}^{j}(\ell)$；

4 $\Phi \leftarrow \{(\ell_1', \ell_2') \in [\![\ell]\!]^2 : \ell_1' + \ell_2' = \ell - \phi(u)\}$ 按 (ℓ_1', ℓ_2') 的字典序排列；

5 $(\ell_1', \ell_2') \leftarrow$ 在 Φ 中最小的数对 $\sum_{(\ell_1', \ell_2') \in \Phi:(\ell_1', \ell_2') \preceq (\ell_1, \ell_2)} M_{j,\ u[\mathrm{key}(j)]}(\ell_1') \cdot W_{\text{下一个}(j)\,i,\ u}(\ell_2') \geq \tau$；

6 $\tau \leftarrow \tau - \sum_{(\ell_1', \ell_2') \in \Phi:(\ell_1', \ell_2') \prec (\ell_1, \ell_2)} M_{j,\ u[\mathrm{key}(j)]}(\ell_1') \cdot W_{\text{下一个}(j)\,i,\ u}(\ell_2')$；

7 $\tau_1 \leftarrow \left\lceil \frac{\tau}{W_{i,u}^{\text{下一个}(j)}(\ell_2)} \right\rceil$，$\tau_2 \leftarrow ((\tau - 1)$ 取模 $W_{\text{下一个}(j)\,i,\ u}(\ell_2)) + 1$；

8 $u_1 \leftarrow$ **RecursiveAccess**$(j, \emptyset, u, \ell_1, \tau_1)$；

9 $u_2 \leftarrow$ **RecursiveAccess**（i，下一个（j）u，ℓ_2，τ_2）；

10 **返回** $u_1 \bowtie u \bowtie u_2$；

连接结果的字典序。这种顺序自然源自连接树 T 中关系的固定遍历顺序，以及每个关系内元组的全序 R_i。

我们提出一个通用过程 **RecursiveAccess**(i, j, v, ℓ, τ)，它返回子树 $\mathrm{T}_i^{\,j}$ 中各关系的连接结果里第 τ 个、分数为 ℓ 且能在其支撑属性上与 v 连接的元组。通过调用 **RecursiveAccess**$(r, \emptyset, \emptyset, \ell, \tau)$，即可解决原始问题。

算法 4 首先确定参与目标连接结果的具体元组 $u \in R_i$。第 1 行对存储在W_{i*}^{j}上的前缀和执行二分搜索，(ℓ)，以定位第一个元组 u：子树 $\mathrm{T}_j^{\,i}$中（分数为 ℓ的）连接结果的累计计数覆盖排名阈值 τ。随后，第 3 行从 τ中减去排在 u之前的元组所贡献的计数。这样，问题就转化为在子树 $\mathrm{T}_j^{\,i}$中检索具体涉及u的（更新后的）第 τ 个连接结果。接下来分两种情况讨论。如果 R_i 是叶节点，则直接返回 u。在一般情况下，根据 $\mathrm{T}_i^{\,j} = \mathrm{T}_j \sqcup \mathrm{T}\ \mathrm{next}(j)\,i$这一事实，我们将检索分解到 T$j$和 T next$(j)\,i$中。目标是确定这些子问题中要查询的局部索引。回顾一下，$W_i^{j}{}_{,u}(\ell)$ 汇总了来自分数组对 (ℓ_1, ℓ_2) 的互不相交的贡献（公式 5）。算法按字典序遍历这些数对（第 4 行），以找到覆盖当前排名阈值 τ的具体数对 (ℓ_1, ℓ_2)（第 5 行）。进一步调整 τ以计入排在前面的数对（第 6 行）后，我们即可将剩余排名映射到 Tj与 T next$(j)\,i$的有效结果笛卡尔积中的索引 τ_1 和 τ_2（第 7 行）。最后，我们在 Tj和 T next$(j)\,i$上递归调用该过程，并合并返回的结果。

完整分析见附录 C。最终，我们得到：

定理 3.3。给定一个大小为 Q 的无环连接实例，以及一组相关的权重函数$\{\mathrm{p}^j\}^{R_j \in Q}$，存在一个大小为 $O(N \log N)$的索引 $\mathcal{D}$，其构建时间为 $O(N \log N \log\log N)$，并且每个子集采样查询都能在 $O(1 + \mu_\Psi \log N)$期望时间内得到回答。

4 连接上的一次性子集采样

我们注意到，定理 3.3 中的数据结构为单次问题提供了一种基线解法。通过构建索引并执行单个查询，我们可以在$O(N\log\ N\log\log\ N+\mu_\Psi\log\ N)$ 期望时间内生成一个子集样本。在本节中，我们将介绍一种适用于单次设置的专用算法，其运行时间为 $O(N\log^2 N+\mu_\Psi)$ 期望时间。当期望样本大小较大（即 $\mu_\Psi \gg N$）时，这种方法优于基线解法，因为它消除了与 μ_Ψ 相关的对数因子。所有省略的证明和细节均见附录 D。

高层思路。 我们首先保留预处理阶段中按照第 3.2 节计算的所有统计量。我们从概念上将连接结果划分为多个桶。回顾一下，回答一次子集采样查询的高层思路是：在实例 $\{\mathrm{B}_\ell,\mathrm{p}:\ \ell\in[\![L-1]\!]\}$ with$p_\ell^+=2^{-\ell}$, and $\bigsqcup_{\ell\geq L}\mathrm{B}_\ell,p$ with $p_L^+=2^{-L}$ 上调用算法 3。每当需要从 B_ℓ 中检索连接结果时，如果$\ell<L$，我们就使用 DirectAccess 预言机来支持检索；否则，我们计算 $\bigsqcup_{\ell\geq L}\mathrm{B}_\ell$ 中的所有结果来支持检索。在前一种情况下，每当在桶 B_ℓ 上以 $\ell<L$ 调用 DirectAccess 预言机时，每次检索都需要花费 $O(\log\ N)$ 时间。为了突破查询时间上的这一瓶颈，我们需要计算额外的统计量。仔细检查算法 4 可知， $O(\log\ N)$ 的开销来自第 1 行的二分搜索和第 5 行的穷举搜索。一个自然的想法是预先计算所有这些统计量。考虑一个任意的关系 $R_i\in Q$及其任意一个子节点 $R_j\in\mathrm{C}_i$。对于每个分数 $\ell\in[\![L-1]\!]$以及每个元组 $\boldsymbol{v}\in R_i[\mathrm{key}(i)]$，我们计算一个大小为 $|R_i\ltimes\boldsymbol{v}|$ 的数组 $X_{i,j,\boldsymbol{v},\ell}$，其中对于每个元组 $\boldsymbol{u}\in R_i\ltimes\boldsymbol{v}$,

$$X_{i,j,\boldsymbol{v},\ell}(\boldsymbol{u})=\sum_{\boldsymbol{u}'\in R_i\ltimes\boldsymbol{v},\,\boldsymbol{u}'\preceq\boldsymbol{u}} W^j_{i,\boldsymbol{u}'}(\ell).$$

接下来，对于每个分数 $\ell\in[\![L-1]\!]$ 和每个元组 $\boldsymbol{u}\in R_i$，我们计算一个大小至多为L的数组 $Y_{i,\ j,\ \boldsymbol{u},\ \ell}$，其中对于每个满足 $\ell_1+\ell_2=\ell-\phi(\boldsymbol{u})$ 的数对 $(\ell_1,\ \ell_2)\in[\![\ell]\!]^2$，按字典序排列，

$$Y_{i,j,\boldsymbol{u},\ell}(\ell_1,\ell_2)=\sum_{(\ell_1',\ell_2')\in[\![\ell]\!]:(\ell_1',\ell_2')\preceq(\ell_1,\ell_2),\ell_1'+\ell_2'=\ell-\phi(\boldsymbol{u})} M_{j,\boldsymbol{u}[\mathrm{key}(j)]}(\ell_1')\cdot W^{\mathrm{next}(j)}_{i,\boldsymbol{u}}(\ell_2').$$

我们注意到，通过构造 X和 Y数组，我们并未为连接上的子集采样定义新的索引。相反，这些统计量是在回答单次子集采样查询时计算的。

算法 4 的每次调用都由一个五元组 $(i,j,\boldsymbol{v},\ell,\tau)$ 标识，该五元组作为其输入。总体思路是维护一个五元组 $(i,j,\boldsymbol{v},\ell,\tau)$ 的列表，并在这些五元组上调用算法 4；同时，通过合并某些操作，将算法 4 的多次调用一并执行。我们按照 $(i,j,\boldsymbol{v},\ell)$ 对这些五元组分组，其中每个分组内的五元组仅有不同的 τ 值，并按 τ 递增排序。我们维护五元组集合 $\mathrm{P}_{i,j}$，以满足算法 4 每次调用的要求。对于节点 R_i 和 $R_j\in\mathrm{C}_i\cup\{\emptyset\}$，$\mathrm{P}_{i,j}$ 定义为算法 4 在 (i,j) 上所有调用的参数集。初始时，对于算法 3 中发出的每个请求 DirectAccess (S_ℓ,τ)，我们将五元组 $(r,\emptyset,\emptyset,\ell,\tau)$ 添加到 $\mathrm{P}_{r,\emptyset}$，其中 R_r 是根节点。下面，我们介绍一个通用过程 BatchRecursiveAccess$(i,j,\mathrm{P}_{i,j})$。新算法执行与算法 4 相同的步骤，但会同时为所有五元组计算所有最小元组 $\boldsymbol{u}$ 和数对 (ℓ_1,ℓ_2)。

BatchRecursiveAccess$(i,j,\mathrm{P}_{i,j})$： 该过程以一个带有子节点$R_j\in\mathrm{C}_i$的节点 $R_i\in Q$ 以及五元组集合 $\mathrm{P}_{i,j}$作为输入。对于每个以五元组 $(i,j,\boldsymbol{v},\ell,\tau)\in\mathrm{P}_{i,j}$为参数的调用，其输出恰好是算法 4 返回的元组集合。伪代码如算法 7 所示。

我们首先根据 τ使用基数排序算法 [19]对 $\mathrm{P}_{i,j}$进行排序。由于 τ在 N中是多项式有界的，使用基数排序可使排序时间减少一个 $\log\ N$的因子。然后，我们按照不同的 $(i,j,\boldsymbol{v},\ell)$ 对 $\mathrm{P}_{i,j}$进行分组，创建分组 $\mathrm{P}_{i,j,\boldsymbol{v},\ell}$。我们注意到，每个 $\mathrm{P}_{i,j,\boldsymbol{v},\ell}$中的所有五元组都依据 $\mathrm{P}_{i,j}$的排序顺序，按照 τ进行了排序。目标是为每个

五元组计算最小元组，如算法 4 第 1 行所示。对于每个分数 $\ell \in [\![L-1]\!]$，我们按照排序顺序考察每个五元组 $(i, j, v, \ell, \tau) \in \mathrm{P}_{i,j,v,\ell}$。我们遍历表 $X_{i,j,v,\ell}$，直到得到第一个满足 $X_{i,j,v,\ell}(\boldsymbol{u}) \geq \tau$ 的元组 $\boldsymbol{u}$。我们还将五元组 (i, j, v, ℓ, τ) 更新为 $(i, j, v, \ell, \tau - X_{i,j,v,\ell}(\mathrm{prev}(\boldsymbol{u})))$，如算法 4 第 3 行所示。如果 R_i 是叶节点，则返回我们从 $\mathrm{P}_{i,j}$中的每个五元组计算出的所有最小元组 $\boldsymbol{u}$。为便于后续引用，我们将五元组 (i, j, v, ℓ, τ) 对应的返回元组 $\boldsymbol{u}$ 记为 $\vec{\boldsymbol{u}}(i, j, v, \ell, \tau)$。

接着，我们以类似的方式计算对 (ℓ_1, ℓ_2)。我们使用基数排序算法，按照 τ 对（已更新的）$\mathrm{P}_{i,j}$进行排序。然后，按照不同的 (i, j, v, ℓ) 对 $\mathrm{P}_{i,j}$进行分组，生成分组 $\mathrm{P}_{i,j,v,\ell}$。目标是像算法 4 第 5 行那样，为每个五元组计算最小的对 (ℓ_1, ℓ_2)。对于每个分数 $\ell \in [\![L-1]\!]$，我们按照排序顺序，依次考虑 $\mathrm{P}_{i,j,v,\ell}$ 中的每个五元组 $(i, j, v, \ell, \tau) \in$。我们遍历表 $Y_{i,j,v,\ell}$，直到找到第一个满足 $Y_{i,j,v,\ell}(\ell_1, \ell_2) \geq \tau$ 的对 (ℓ_1, ℓ_2)。我们按照算法 4 第 7 行的方式计算 τ_1, τ_2。此外，我们将五元组 $(j, \emptyset, \boldsymbol{u}, \ell_1, \tau_1)$ 添加到（初始为空的）集合 $\mathrm{P}_{j,\emptyset}$中，并将五元组 $(i, \mathrm{next}(j)\ \boldsymbol{u}, \ell_2, \tau_2)$ 添加到（初始为空的）集合 $\mathrm{P}_{i,\mathrm{next}(j)}$中。注意，这些集合将作为下一次调用 BatchRecursiveAccess$(j, \emptyset, \mathrm{P}_{j,\emptyset})$ 和 BatchRecursiveAccess$(i, \mathrm{next}(j), \mathrm{P}_{i,\mathrm{next}(j)})$ 的输入，与算法 4 第 8 行和第 9 行相同。

现在，我们将为 $\mathrm{P}_{i,j}$ 中的五元组生成要返回的元组。假设两个递归调用返回的元组集合已知。对于每个五元组 $(i, j, v, \ell, \tau) \in \mathrm{P}_{i,j}$，令 $\vec{\boldsymbol{u}}(j, \emptyset, \boldsymbol{u}, \ell_1, \tau_1)$ 为五元组 $(j, \emptyset, \boldsymbol{u}, \ell_1, \tau_1) \in \mathrm{P}_{j,\emptyset}$ 返回的元组，令 $\vec{\boldsymbol{u}}(i, \mathrm{next}(j)\ \boldsymbol{u}, \ell_2, \tau_2)$ 为五元组 $(i, \mathrm{next}(j)\ \boldsymbol{u}, \ell_1, \tau_1) \in \mathrm{P}_{i,\mathrm{next}(j)}$ 返回的元组。我们只需将元组 $\vec{\boldsymbol{u}}(j, \emptyset, \boldsymbol{u}, \ell_1, \tau_1)$、元组 $\vec{\boldsymbol{u}}(i, \mathrm{next}(j)\ \boldsymbol{u}, \ell_2, \tau_2)$ 与元组 $\boldsymbol{u}$ 连接起来，构成元组 $\vec{\boldsymbol{u}}(i, j, \boldsymbol{u}, \ell, \tau)$。最后，我们将返回 $\mathrm{P}_{i,j}$ 中每个五元组 (i, j, v, ℓ, τ) 对应的所有元组。

定理 4.1。对于大小为 N 的无环连接实例 Q，以及一组关联的权重函数$\{\mathrm{p}^j\}^{R_j \in Q}$，存在一种算法，能够在$O\left(\min\left\{N \log N \log\log N + \mu_\Psi \log N,\ N \log^2 N + \mu_\Psi\right\}\right)$期望时间内，返回该实例的一个子集样本$\langle \mathrm{Join}(Q), \mathrm{p}\rangle$ $\Psi =$。

5 连接上的动态子集采样

本节转向动态设置，其中元组逐个插入。我们首先调整 [22] 的技术，直接将定理 3.2 中的非最优（静态）索引扩展为动态变体，以维护一个近似 DirectAccess 索引。然后，我们引入新的技术和思路，将定理 3.3 中优化后的静态索引扩展到动态设置。全文中，我们假设每个关系中的元组都按其插入时间戳排序。

5.1 首次尝试

总体思路是近似维护第 3 节中提出的静态索引。然而，正如 [22],所指出的，即使只是维护三元连接 $Q = \{R_1(X, Y)\ R_2(Y\ \ Z)\ R_3(Z,\ W)\}$ 的连接大小，每次插入也至少需要 $\Omega(\sqrt{N})$ 时间，这在流式场景中代价过高。因此，我们采用 [22] 提出的方法，更高效地维护近似 DirectAccess 索引，同时渐近地不牺牲整体抽样效率。我们首先定义增量查询 $\Delta\mathrm{Join}(Q, \boldsymbol{u})$，它表示将元组 $\boldsymbol{u}$ 插入关系 Q时产生的新连接结果集合。形式化地，如果将 $\boldsymbol{u}$ 插入 R_i，则 $\Delta\mathrm{Join}(Q,\ \boldsymbol{u}) = \mathrm{Join}(Q \setminus \{R_i\} \cup \{R_i \cup \{\boldsymbol{u}\}\}) \setminus \mathrm{Join}(Q)$。

定理 5.1 ([22])。给定一个初始为空的无环连接 Q，我们可以维护一个索引 $\mathcal{L}$，它位于 Q 上并支持以下操作：

- 当一个元组 t被添加到 Q时，可以在摊销 $O\mathcal{L}(\log N)$ 时间内更新。

• 该索引定义了一个数组 $J \supseteq$ 连接(Q)，其中 连接(Q) 中的元组是真实元组，其余则是虚拟元组。该索引可以在 $O(1)$ 时间内返回 $|J|$。对于任意给定的 $j \in [|J|]$，它可以在 $O(\log N)$ 时间内返回 $J[j]$。此外，可以保证 J 是 λ-稠密的，且其概率为某个常数 $0 < \gamma \leq 1$。
• 将 Join(Q) 替换为任意 $\boldsymbol{u} \notin Q$ 的增量查询 $\Delta\text{Join}(Q, \boldsymbol{u})$ 时，上述结论同样成立。其中 N 为插入总数，但索引无需提前获知 N。

利用定理 5.1，我们通过改编第 3.1 节中的框架，推导出一个简单的动态算法。与第 3.1 节一样，我们定义 $(L+1)^k$ 个子实例 Q_j，其中 $\boldsymbol{j} \in [\![L]\!]^k$。在动态设置下，我们为每个子实例 Q_j维护一个动态索引 Lj （如定理 5.1 所述）。将元组 t 插入关系 R_i时，我们首先确定其桶索引 a，使得 $t \in R_i^{\langle a \rangle}$。该元组 t 参与 $(L+1)^{k-1}$ 个子实例。我们更新相应的 $(L+1)^{k-1}$ 个索引。由于每次更新的摊销时间为 $O(\log N)$，总摊销更新时间为$O(L^{k-1} \log N)$。为回答子集采样查询，我们采用批量拒绝采样策略。对于每个子实例 Q_j，索引 Lj 会给出隐式数组 $|J_j|$的大小。我们用 $|J_j|$ 作为子连接大小的上界，以定义元采样概率 q$(\boldsymbol{j})$。假设选中了一个子实例 Q_j，并且算法需要获取元组 $u = J_j[z]$。如果 u是一个虚拟元组（这种情况发生的概率至多为 $1-\gamma$），我们就拒绝它。如果 u是真实元组，则按照其权重执行标准的拒绝检查。由于 γ是常数，期望查询时间在渐近意义上仍与静态情形相同。

推论 5.2。给定一个无环连接实例 Q以及一组相关联的权重函数 $\{\mathrm{p}j\}_{R_j \in Q}$，我们可以在元组插入时维护一个动态索引，使用 $O(N \log^{k-1} N)$ 空间，摊销更新时间为 $O(\log^k N)$，同时支持在 $O(1 + \mu_\Psi \log N)$期望时间内回答子集采样查询，其中 N 是插入总数。

5.2 连接上的子集采样优化动态索引

为了同时改善空间和更新时间，我们将第 3.2 节中的优化索引改造为适用于动态设置。由于第 3.2 节中的索引本质上是静态的，因此这种改造并非易事，需要采用新的思路。我们不再为每种桶组合维护互不相交的索引，而是直接维护聚合统计量（即W和 M值）。由于维护精确计数的代价过高，我们改为存储向上取整至下一个 2 的幂的近似统计量。

对于每个关系 $R_i \in Q$、每个子节点 $R_j \in \mathsf{C}_i$ 以及每个元组 $\boldsymbol{u} \in R_i$，我们维护 $\tilde{W}_i^j{}_{,\boldsymbol{u}}(\ell)$ 在 $W_i^j{}_{,\boldsymbol{u}}(\ell)$ 上的上界。类似地，对于每个非根节点 R_i、元组 $\boldsymbol{v} \in R_i[\text{key}(i)]$, 和分数 $\ell \in [\![L-1]\!]$，我们将 $\hat{}M_{i,\boldsymbol{v}}(\ell)$ 维护为以下近似计数器之和：

$$\hat{M}_{i,\boldsymbol{v}}(\ell) = \sum_{\boldsymbol{u} \in R_i \ltimes \boldsymbol{v}} \tilde{W}^{\emptyset}_{i,\boldsymbol{u}}(\ell), \tag{6}$$

并使用它计算上界 $\tilde{M}_{i,\boldsymbol{v}}(\ell) = 2^{\lceil \log \hat{}M_{i,\boldsymbol{v}}(\ell) \rceil}$。考虑一个叶节点 R_i。由于 $\mathsf{C}_i = \emptyset$，当 $\ell = \phi(\boldsymbol{u})$ 时，$\tilde{W}_i^{\emptyset}{}_{,\boldsymbol{u}}(\ell) = 1$；否则为 $\tilde{W}_i^{\emptyset}{}_{,\boldsymbol{u}}(\ell) = 0$。考虑一个内部节点 R_i。假设已为每个子节点 $R_j \in \mathsf{C}_i$ 计算出 $\tilde{W}$ 和 $\tilde{M}$ 值。我们按照节点在 C_i 中的递减顺序计算 $\tilde{W}_i^j{}_{,\boldsymbol{u}}$。令 $R_{\text{next}(j)} \in \mathsf{C}_i$ 为 R_j 的紧邻兄弟节点。

$$\tilde{W}^j_{i,\boldsymbol{u}}(\ell) = \sum_{(\ell_1,\ell_2) \in [\![\ell - \phi(\boldsymbol{u})]\!] : \ell_1 + \ell_2 = \ell - \phi(\boldsymbol{u})} \tilde{M}_{j,\boldsymbol{u}[\text{key}(j)]}(\ell_1) \cdot \tilde{W}^{\text{next}(j)}_{i,\boldsymbol{u}}(\ell_2). \tag{7}$$

此外，我们定义 $\tilde{W}\text{next}_{(j^*)\,i,\boldsymbol{u}}(0) = 1$和 $\tilde{W}\text{next}_{(j^*)\,i,\boldsymbol{u}}(\ell) = 0$，用于 $\ell > 0$最大的节点 $R_{j^*} \in \mathsf{C}_i$。

[4]如果对于所有 i，都有 $r_i \geq \lambda \cdot (i-1)$，则数据流 $S = x_1, x_2, \ldots\ldots, x_n$是 λ-稠密的，其中 r_i 表示前 $i-1$ 个元素中的真实（非虚拟）元素数量。

算法 5：更新$(i,\ \boldsymbol{v},\ \ell,\ \Delta)$

输入： $i \in [k]$，元组 $\boldsymbol{v} \in \mathbf{dom}(\mathrm{key}(i))$，分数 ℓ，以及 $\tilde{W}_i^{\emptyset}[\boldsymbol{u}](\ell)$ 增加了 Δ，用于 $\boldsymbol{u} \in R_i \ltimes \boldsymbol{v}$。

更新为 $\{\tilde{W}_i^{\emptyset}$ 构建的前缀和树 $_{,\boldsymbol{u}'}(\ell) :\ \boldsymbol{u}' \in R_i, \boldsymbol{u}'[\mathrm{key}(i)] = \boldsymbol{u}[\mathrm{key}(i)]\}$;

$\hat{M}_{i,\boldsymbol{v}}(\ell) \leftarrow \hat{M}_{i,\boldsymbol{v}}(\ell) + \Delta$;

$\tilde{M}_{i,\boldsymbol{v}}(\ell) \leftarrow 2^{\lceil \log \hat{M}_{i,\boldsymbol{v}}(\ell) \rceil}$;

if $\tilde{M}_{i,\ \boldsymbol{v}}(\ell)$ 更改和父节点$(i) \neq \mathrm{null}$ **那么**

  $p \leftarrow$父节点(i);

  对于每个元组 $\boldsymbol{u} \in R_p \ltimes \boldsymbol{v}$ **执行**

    根据（7）为每个分数 $\ell' \in [\![L-1]\!]$ 和 $R_j \in \mathsf{C}_p$ 计算 $\tilde{W}_{p,\ \boldsymbol{u}}^{j}(\ell')$;

    对每个 $\ell' \in [\![L-1]\!]$ **执行**

      如果 $\tilde{W}_{p,\ \boldsymbol{u}(\ell')}^{\emptyset}$ 增加了（增加量为 Δ'），**则**

        更新$(p,\ \boldsymbol{u}[\text{键}(p)],\ \ell',\ \Delta')$;

我们指出，$\tilde{W}_{i,\ \boldsymbol{u}}^{j}(\ell)$（分别为 $\tilde{M}_{i,\ \boldsymbol{v}}(\ell)$）是 $W_{i,\ \boldsymbol{u}}^{j}(\ell)$（分别为 $M_{i,\ \boldsymbol{v}}(\ell)$）的常数近似。对于每个分数 $\ell \in [\![L-1]\!]$和每个子节点 $R_j \in \mathsf{C}_i$，我们按照预先确定的顺序，为所有元组 $\boldsymbol{u} \in R_i$ 上的值 $\tilde{W}_{i,\ \boldsymbol{u}}^{j}(\ell)$ 维护一棵动态前缀和树。这样我们便能在 $O(\log N)$ 时间内支持单点更新和前缀和查询。

更新过程。当一个新元组 $\boldsymbol{u}$ 插入关系 R_i 时，我们首先初始化其局部统计量。如果 R_i 是叶节点，则 $\tilde{W}_i^{\emptyset}[\boldsymbol{u}](\ell) = 1$；如果 $\ell = \phi(\boldsymbol{u})$ 且 $\tilde{W}_i^{\emptyset}[\boldsymbol{u}](\ell) = 0$，则另作处理。如果 R_i 是内部节点，我们通过公式（7）计算所有分数 $\ell \in [\![L-1]\!]$ 对应的 $\tilde{W}_i^{\emptyset}[\boldsymbol{u}](\ell)$，使用快速傅里叶变换耗时 $O(L \log L)$。初始化 $\boldsymbol{u}$ 后，我们必须更新父节点中的统计量。具体来说，插入 $\boldsymbol{u}$ 会使 $\boldsymbol{v} = \boldsymbol{u}[\mathrm{key}(i)]$ 的 $\hat{M}_{i,\boldsymbol{v}}(\ell) = \sum_{\boldsymbol{u}' \in R_i \ltimes \boldsymbol{v}} \tilde{W}_i^{\emptyset}[\boldsymbol{u}'](\ell)$ 增加。我们更新 $\hat{M}_{i,\boldsymbol{v}}(\ell)$，并检查 $M_{i,\boldsymbol{v}}(\ell)$ 是否发生变化。如果 $M_{i,\boldsymbol{v}}(\ell)$ 发生变化，我们必须将此变化传播到父关系 $R_{\mathrm{parent}(i)}$。算法 5 描述了这一传播过程。当一个元组 $\boldsymbol{u}$ 插入叶节点 R_i 时，我们只需调用 $\mathrm{Update}(i, \boldsymbol{u}[\mathrm{key}(i)], \phi(\boldsymbol{u}), 1)$。如果它被插入内部节点，则对每个 $\ell \in [\![L-1]\!]$ 调用 $\mathrm{Update}(i, \boldsymbol{u}[\mathrm{key}(i)], \ell, \tilde{W}_i^{\emptyset}[\boldsymbol{u}](\ell))$。

复杂度分析。更新开销主要由变更的传播决定。注意，对于任意固定的 $i, \boldsymbol{v}, \ell$，

$\tilde{M}_{i,\boldsymbol{v}}(\ell)$ 是一个上界，最多增加 $O(\log N)$ 次，因为只有当 $\hat{M}_{i,\boldsymbol{v}}(\ell)$ 翻倍时，$M_{i,\boldsymbol{v}}(\ell)$ 才会增加。当 $\tilde{M}_{i,\boldsymbol{v}}(\ell)$ 发生变化时，我们会对父节点中可与 $\boldsymbol{v}$ 连接的元组执行卷积运算。由于 $L = O(\log N)$，卷积运算需要 $O(L \log L)$ 时间。我们证明，在长度为 N 的整个数据流中，M 值变化的总次数有界为 $O(NL \log N)$。因此，摊销更新时间为 $O(L^2 \log N \log L) = O(\log^3 N \log\log N)$。空间复杂度仍为 $O(NL) = O(N \log N)$。

查询过程。查询过程遵循第 3.2 节中的优化策略，在得分桶 B_ℓ（针对 $\ell < L$）和尾部桶 $\mathrm{B}_{\geq L}$上调用算法 3。为了从选定的桶中取回第 τ 个元组，我们采用算法 4 的修改版本，该版本会遍历近似的 $\tilde{W}$ $\tilde{M}$统计量W和 M。如第 5.1 节所述，由于这些统计量是上界，索引隐式地定义了一个包含“虚拟”元组的连接结果超集。如果取回的元组是一个虚拟元组，我们就拒绝它。由于W是 W 的常数因子近似，因此抽中真实元组的概率至少为某个常数 c 的 2^{-c}。因此，抽取 χ 个元组需要 $O(\chi)$ 期望时间，整体期望查询时间仍为 $O(1 + \mu_\Psi \log N)$。

我们陈述如下定理，具体细节见附录 F。

定理 5.3。 给定一个无环连接实例 Q和一组相关的权重函数 $\{\mathrm{p}_j\}^{R_j \in Q}$，我们可以在元组插入的情况下维护一个动态索引，使用 $O(N$对数 $N)$空间，摊销更新时间为 $O($对数$^3\, N$对数的对数 $N)$，同时支持在 $O(1+\mu_\Psi$对数 $N)$期望时间内回答子集采样查询，其中 N 为插入总次数。

此索引可用于生成一次性子集样本，运行时间如下：

推论 5.4。 给定一个无环连接实例 Q及一组相关的权重函数 $\{\mathrm{p}_j\}^{R_j \in Q}$，我们可以在 $O(N \log^3 N \log\log N + \mu_\Psi \log N)$期望时间内维护一个一次性子集样本，其中$N$为插入总数。

6 结论

我们提出了首个无需完全物化、用于连接上的子集采样的高效框架。通过利用查询的连接结构和权重函数的可分解性，我们的算法在静态索引、一次性采样以及插入操作下的动态维护方面实现了近乎最优的复杂度。这项工作为未来研究开辟了若干引人入胜的方向。我们希望将理论框架扩展到支持更广泛的权重定义。本文讨论了标准聚合函数，但如何在复杂、非单调或整体聚合函数下实现高效采样，仍是一个开放问题。此外，我们计划通过在真实世界的关系数据机器学习工作负载上实现并评估索引，弥合理论与实践之间的差距。

参考文献

[1] S. Abiteboul、R. Hull 和 V. Vianu。《数据库基础》，第 8 卷。Addison-Wesley Reading，1995 年。[2] M. Abo-Khamis、S. Im、B. Moseley、K. Pruhs 和 A. Samadian。用于支持向量机训练的关系梯度下降算法。载于计算机系统算法原理研讨会（APOCS），第 100–113 页。SIAM，2021 年。[3] P. K. Agarwal、A. Esmailpour、X. Hu、S. Sintos 和 J. Yang。计算合取查询结果的良好代表性摘要。ACM数据管理会议录，2(5):1–27，2024 年。[4] M. Arenas、T. C. Merkl、R. Pichler 和 C. Riveros。迈向查询答案多样性的可处理性：超度量来助力。ACM数据管理会议录，2(5):1–26，2024 年。[5] A. Atserias、M. Grohe 和 D. Marx。关系连接的大小界与查询计划。SIAM计算杂志，42 (4):1737–1767，2013 年。[6] O. Bachem、M. Lucic 和 A. Krause。面向机器学习的实用核心集构造。arXiv 预印本 arXiv:1703.06476，2017 年。[7] C. Berkholz、J. Keppeler 和 N. Schweikardt。更新条件下的合取查询回答。载于第 36 届 ACM SIGMOD-SIGACT-SIGAI数据库系统原理研讨会论文集，PODS '17，第 303–318 页，美国纽约州纽约，2017 年。计算机协会。ISBN 9781450341981。[8] S. Bhattacharya、P. Kiss、A. Sidford 和 D. Wajc。二分图分数匹配的近乎最优动态舍入。载于ACM计算理论研讨会（STOC）论文集，第 59–70 页。ACM，2024 年。[9] A. Borodin 和 J. I. Munro。代数问题与数值问题的计算复杂性，第 1 卷。爱思唯尔，1975 年。[10] K. Bringmann 和 K. Panagiotou。离散分布的高效采样方法。载于自动机、语言与编程国际研讨会（ICALP）论文集，第 133–144 页，2012 年。[11] N. Carmeli、S. Zeevi、C. Berkholz、B. Kimelfeld 和 N. Schweikardt。使用随机访问和随机顺序枚举回答（并集）合取查询。载于第 39 届 ACM SIGMOD-SIGACT-SIGAI数据库系统原理研讨会论文集，第 393–409 页，2020 年。[12] N. Carmeli、N. Tziavelis、W. Gatterbauer、B. Kimelfeld 和 M. Riedewald。实现合取查询排名答案直接访问的可处理顺序。ACM数据库系统汇刊，48(1):1–45，2023 年。[13] S. Chaudhuri、R. Motwani 和 V. Narasayya。连接上的随机采样。载于 1999 年 ACM SIGMOD数据管理国际会议论文集，SIGMOD '99，第 263–274 页，美国纽约州纽约，1999 年。计算机协会。ISBN 1581130848。[14] J. Chen、Q. Yang、R. Huang 和 H. Ding。关系数据核心集及其应用。神经信息处理系统进展，35:434–448，2022 年。[15] Y. Chen 和 K. Yi。循环连接上的随机采样与大小估计。载于 C. Lutz 和 J. C. Jung 编，第二十三届数据库理论国际会议（ICDT 2020），莱布尼茨信息学国际会议录（LIPIcs）第 155 卷，第 7:1–7:18 页，德国达格施图尔，2020 年。达格施图尔城堡——莱布尼茨信息学中心。ISBN 978-3-95977-139-9。[16] Z. Cheng 和 N. Koudas。规范化数据上的非线性模型。载于 2019 年 IEEE 第 35 届数据工程国际会议（ICDE），第 1574–1577 页。IEEE，2019 年。[17] Z. Cheng、N. Koudas、Z. Zhang 和 X. Yu。规范化数据上非线性模型的高效构建。载于 2021 年 IEEE 第 37 届数据工程国际会议（ICDE），第 1140–1151 页。IEEE，2021 年。[18] J. W. Cooley 和 J. W. Tukey。用于机器计算复傅里叶级数的算法。计算数学，19(90):297–301，1965 年。[19] T. H. Cormen、C. E. Leiserson、R. L. Rivest 和 C. Stein。《算法导论》。麻省理工学院出版社，2022 年。[20] G. Cormode、M. Garofalakis、P. J. Haas 和 C. Jermaine。海量数据的概要：样本、直方图、小波与草图。《数据库基础与趋势》，4(1–3):1–294，2012 年。[21] R. Curtin、B. Moseley、H. Ngo、X. Nguyen、D. Olteanu 和 M. Schleich。Rk-means：面向关系数据的快速聚类。载于人工智能与统计国际会议论文集，第 2742–2752 页。PMLR，2020 年。[22] B. Dai、X. Hu 和 K. Yi。连接上的蓄水池采样。载于 ACM数据管理会议（SIGMOD）论文集，第 1–26 页，2024 年。[23] S. Deng、S. Lu 和 Y. Tao。论连接采样与组合输出敏感连接算法的困难性。载于第 42 届 ACM SIGMOD-SIGACT-SIGAI数据库系统原理研讨会论文集，PODS '23，第 99–111 页，美国纽约州纽约，2023 年。计算机协会。ISBN 9798400701276。[24] N. Duffield、C. Lund 和 M. Thorup。用于任意子集和估计的优先级采样。ACM学报（JACM），54(6):32，2007 年。[25] A. Esmailpour 和 S. Sintos。关系聚类的改进近似算法。ACM数据管理会议录，2(5):1–27，2024 年。[26] D. Feldman 和 M. Langberg。用于数据近似与聚类的统一框架。载于ACM计算理论研讨会（STOC）论文集，第 569–578 页，2011 年。

[27] G. Gottlob、G. Greco、F. Scarcello 等。树宽与超树宽。可处理性：应对难题的实用方法，1:20，2014 年。[28] P. J. Haas 和 J. M. Hellerstein。用于在线聚合的涟漪连接。载于*1999* 年 *ACM SIGMOD*数据管理国际会议论文集，SIGMOD '99，第 287–298 页，纽约，纽约州，美国，1999 年。计算机协会。ISBN 1581130848。[29] J. Huang 和 S. Wang。子集采样及其扩展。*arXiv* 预印本 *arXiv:2003.01075*，2023 年。[30] A. Kara、M. Nikolic、D. Olteanu 和 H. Zhang。F-ivm：更新条件下关系数据库上的分析。*VLDB Journal*，33(4)：903–929，2024 年。[31] M. A. Khamis、H. Q. Ngo、X. Nguyen、D. Olteanu 和 M. Schleich。Ac/dc：数据库内机器学习的惊人突破。载于第二届端到端机器学习数据管理研讨会论文集，第 1–10 页，2018 年。[32] K. Kim、J. Ha、G. Fletcher 和 W.-S. Han。为连接上的均匀分布抽样和规模估计保证 Õ(agm/out) 运行时间。载于第 *42* 届 *ACM SIGMOD-SIGACT-SIGAI*数据库系统原理研讨会论文集，PODS '23，第 113–125 页，纽约，纽约州，美国，2023 年。计算机协会。ISBN 9798400701276。[33] A. Kumar、J. Naughton 和 J. M. Patel。在规范化数据上学习广义线性模型。载于*ACM*数据管理会议（*SIGMOD*）论文集，第 1969–1984 页，2015 年。[34] F. Li、B. Wu、K. Yi 和 Z. Zhao。漫游连接：通过随机游走实现在线聚合。载于*2016* 年数据管理国际会议论文集，第 615–629 页，2016 年。[35] T. C. Merkl、R. Pichler 和 S. Skritek。合取查询的答案多样性。计算机科学逻辑方法，21，2025 年。[36] B. Moseley、K. Pruhs、A. Samadian 和 Y. Wang。用于 k-means 聚类的关系算法。载于自动机、语言与编程国际讨论会，2021 年。[37] R. Motwani 和 P. Raghavan。随机化算法。剑桥大学出版社，1995 年。[38] H. Q. Ngo。最坏情况最优连接算法：技术、成果与开放问题。载于*ACM*数据库系统原理研讨会（*PODS*）论文集，第 111–124 页。ACM，2018 年。[39] F. Olken。从数据库中随机抽样。博士论文，加州大学伯克利分校，1993 年。[40] F. P. Preparata 和 M. I. Shamos。计算几何。计算机科学文本与专著。柏林，施普林格出版社，1985 年。[41] M. Schleich、D. Olteanu 和 R. Ciucanu。在因式分解连接上学习线性回归模型。载于*ACM*数据管理会议（*SIGMOD*）论文集，第 3–18 页，2016 年。[42] M. Schleich、D. Olteanu、M. Abo-Khamis、H. Q. Ngo 和 X. Nguyen。关系数据上的模型学习：简要教程。载于可扩展不确定性管理：第 *13* 届国际会议，*SUM 2019*，第 423–432 页。施普林格，2019 年。[43] K. Yang、Y. Gao、L. Liang、B. Yao、S. Wen 和 G. Chen。面向在规范化数据上使用高斯核的因式分解 SVM。载于*2020* 年 *IEEE* 第 *36* 届数据工程国际会议（*ICDE*），第 1453–1464 页。IEEE，2020 年。[44] M. Yannakakis。无环数据库模式的算法。载于第七届超大型数据库国际会议论文集——第 *7* 卷，VLDB '81，第 82–94 页。VLDB 基金会，1981 年。[45] L. Yi、H. Wang 和 Z. Wei。最优动态子集采样：理论与应用。载于*ACM*知识发现与数据挖掘会议（*SIGKDD*）论文集，第 3116–3127 页。ACM，2023 年。[46] Z. Zhao、R. Christensen、F. Li、X. Hu 和 K. Yi。重新审视连接上的随机采样。载于*ACM*数据管理会议（*SIGMOD*）论文集，2018 年。

完整符号表

	符号	描述
General	S	一组元素。
	p	一个为 S 中的每个元素分配概率的概率函数。
	$\Psi = S$,p	一个子集采样问题实例。
	$\mathbf{X}$	来自子集采样实例的一个随机子集样本。
	μ_Ψ	子集样本$\mathbf{X}$ 的期望大小来自 Ψ。
	$[x]$	整数集 {1、2、.....、 x}。
	$[\![x]\!]$	整数集 {0、1、2、.....、 x}。
	直接访问	一个直接访问预言机。
Subset Sampling over Joins	**att**, **dom**	所有属性的集合以及所有值的域。
	$\boldsymbol{u}, \boldsymbol{v}$	元组。
	R_i	连接查询中的第 i个关系。
	模式(R_i)	关系 R_i 的模式（属性集合）。
	$Q = \{R_1, \ldots, R_k\}$	由 k 个关系组成的连接查询。
	N	Q 的输入大小，即 $\sum_{i=1}^{k} \lvert R_i \rvert$。
	连接(Q)	连接查询 Q 的结果元组集合。
	$G = (V, E)$	连接查询 Q的模式超图。
	ρ^*	模式图 G的分数边覆盖数。
	$\mathrm{p}_i(\boldsymbol{u})$	与元组 $\boldsymbol{u} \in R_i$ 相关的概率。
	p($\boldsymbol{u}$)	连接结果 $\boldsymbol{u}$ 的概率，定义为 $\prod_{i=1}^{k} \mathrm{p}_i(\boldsymbol{u}[\text{模式}(R_i)])$。
	L	分区/桶的数量，定义为 $\lceil \rho^* \log N \rceil$。
	$\phi(\boldsymbol{u})$	元组 $\boldsymbol{u}$ 的分数。
	$\mathcal{B}_\ell$	分数为 ℓ 的连接结果集合（桶）。
Optimized Index	$\mathcal{T}$	无环连接的连接树。
	$\mathcal{T}_i$	以关系 R_i 为根节点的 T 的子树。
	C_i	T 中关系 R_i 的子节点集合。
	$\mathcal{T}_i^j$	T$_i$ 的部分子树，不包括位于 R_j 之前的子节点。
	key(i)	R_i 与其在 T 中的父节点之间的公共连接属性。
	W_i^j	T$_i^j$ 中涉及 $\boldsymbol{u}$ 且分数为 ℓ 的连接结果数量。
	$M_{i,\boldsymbol{v}}(\ell)$	T的聚合计数 -值，适用于投影到 $\boldsymbol{v}$的元组。
Dynamic	η	流式场景中的时间戳。
	Q^η	由数据流中前 η 个元组定义的连接。
	p$^\eta$	与 Q^η关联的权重函数。
	$\tilde{W}_i^j$	W_i^j 的近似上界
	$\tilde{M}_{i,\boldsymbol{v}}(\ell)$	M_i 的近似上界，$\boldsymbol{v}(\ell)$。 $'_{\boldsymbol{u}}(\ell)$。

表 2. 符号表

B 第 2 节缺失材料

引理 2.4 的证明。预处理阶段构建两级索引：为每个子实例 Ψ_i 构建标准子集采样索引（耗时 $O(\sum |S_i|) = O(|S|)$），并为 $\Psi_{\text{meta}} = [m]$，q 构建元索引（耗时 $O(m) \le O(|S|)$）。空间复杂度显然为 $O(|S|)$。

现在分析算法 3 的期望查询时间。运行时间由两部分组成：**元采样：**使用元索引生成索引集合 $\mathbf{I}$，期望耗时为 $O(1+\mu_{\Psi_{\text{meta}}})$，其中 $\mu_{\Psi_{\text{meta}}} = \sum_{i=1}^{m} \mathrm{q}(i)$。

子采样：对于每个选定的索引 $i \in \mathbf{I}$，我们使用几何跳跃过程（模拟 **uss-advanced**）生成一个中间样本 $\mathbf{X}'_i$。对于特定的 i，其成本与该中间样本的大小成正比。总期望成本为：

$$\mathbf{E}[\text{Cost}_{\text{sub}}] = \sum_{i=1}^{m} \mathrm{q}(i) \cdot O\left(\mathbf{E}[|\mathbf{X}'_i| \mid \mathbf{X}'_i \neq \emptyset]\right).$$

注意，$\mathbf{E}[|\mathbf{X}'_i| \mid \mathbf{X}'_i \neq \emptyset] = \frac{\mathbf{E}[|\mathbf{X}'_i|]}{\Pr[\mathbf{X}'_i \neq \emptyset]} = \frac{|S_i| p_i^+}{\Pr[\mathbf{X}'_i \neq \emptyset]}$。将其代回求和中，$\mathrm{q}(i)$ 项相互抵消：

$$\mathbf{E}[\text{Cost}_{\text{sub}}] = O\left(\sum_{i=1}^{m} \mathrm{q}(i) \cdot \frac{|S_i| p_i^+}{\mathrm{q}(i)}\right) = O\left(\sum_{i=1}^{m} |S_i| p_i^+\right).$$

合并两部分后，总期望时间为 $O(1+\sum_{i=1}^{m} \mathrm{q}(i)+\sum_{i=1}^{m} |S_i| p_i^+)$。由于 $\mathrm{q}(i) \le |S_i| p_i^+$，总复杂度可简化为 $O(1 + \sum_{i=1}^{m} |S_i| p_i^+)$。

最后，如果每个 Ψ_i 都是 β-均匀分布的，则有 $p_i^+ \le \beta \cdot \min_{e \in S_i} \mathrm{p}_i(e)$。对 S_i 中的所有元素求和，得到 $|S_i| p_i^+ \le \beta \sum_{e \in S_i} \mathrm{p}_i(e) = \beta \mu_{\Psi_i}$。对所有 i 求和后，总成本有界于$O(1 + \sum_i \beta \mu_{\Psi_i}) = O(1 + \beta \mu_{\Psi})$。□

C 第 3 节缺失材料

C.1 高层框架分析

引理 3.1 的证明。对于最大概率，我们始终有 $\max_{\boldsymbol{u} \in \text{Join}(Q_j)} \mathrm{p}(\boldsymbol{u}) \le 2^{-\sum_{i \in [k]} j_i}$。若 $\sum_{i \in [k]} j_i \ge L$，则根据我们对 L 的假设，有 $\max_{\boldsymbol{u} \in \text{Join}(Q_j)} \mathrm{p}(\boldsymbol{u}) \le 2^{-\sum_{i \in [k]} j_i} \le 2^{-L} \le 1/|\text{连接}(Q)|$。因此，$\Psi_j$ 是轻量的。否则，必有 $L \notin \{j_1, j_2, \ldots, j_k\}$。此时，对于最小概率，我们有 $2^{-k-\sum_{i \in [k]} j_i} \le \min_{\boldsymbol{u} \in \text{Join}(Q_j)} \mathrm{p}(\boldsymbol{u})$。因此，$\Psi_j$ 是 2^k-均匀分布的，因为 $\max_{\boldsymbol{u} \in \text{Join}(Q_j)} \mathrm{p}(\boldsymbol{u}) \le \min_{\boldsymbol{u} \in \text{Join}(Q_j)} \mathrm{p}(\boldsymbol{u})$ □

定理 3.2 的证明。通过分析采样策略的预处理成本和查询时间即可得证。

预处理与数据结构。 预处理阶段构建引理 2.4 中定义的复合索引。

(1) 划分与子索引。 我们将每个关系 $R_i \in Q$ 划分为 $L+1$ 个子关系，从而定义 $(L+1)^k$ 个子实例 Q_j。对于每个子实例，我们构建一个 DirectAccess 预言机（子索引）。根据 [46],，为无环连接构建 DirectAccess 预言机所需时间与其输入大小呈线性关系。由于每个元组参与 $(L+1)^{k-1}$ 个子实例，且 $L = O(\log N)$，总时间和空间为 $O(N \log^{k-1} N)$。**(2) 元索引。** 我们为 $\Psi' = [\![L]\!]^k$，q构建元索引。由于论域大小为 $(L+1)^k = O(\log^k N)$，该索引可在 $O(\log^k N)$ 时间和空间内构建，这一成本可由构建子索引的成本涵盖。

查询求解。 该查询按照算法 3 执行。

(1) 阶段 1：元采样。 我们查询元索引以获取索引 $\mathbf{I}$。期望时间为$O(1+\mu_{\Psi'})$，其中 $\mu_{\Psi'}=\sum_j \mathsf{q}(j)$。如正文所示，$\mu_{\Psi'}\le \mu_{\Psi}+1$。
(2) 阶段 2：子采样。 对于每个 $j\in\mathbf{I}$，我们从 Q_j中抽取样本。由于每个子实例要么是轻量的，要么是 2^k-均匀分布的，因此每个选定实例的预期成本与样本数量乘以直接访问的访问成本 $O(\log N)$ 成正比。根据引理 2.4 中的分析，总预期成本为 $O((1+\mu_{\Psi})\log N)$。

将这些相加，得到总预期查询时间 $O(1+\mu_{\Psi}\log N)$。 □

C.2 优化算法的预处理

我们将说明如何计算用于构建所需索引的统计量。完整的伪代码见算法 6。下面我们将详细说明其直观思路。我们的预处理阶段以自底向上的方式进行计算。注意，M-值定义在W-值之上，因此对于每个节点 $R_i\in Q$，一旦W-值计算完成，就可以直接根据 (4) 计算 M-值。下面我们重点介绍W-值的计算。

考虑叶节点 R_i。根据 (3) 的定义，我们有 $W_i^{\emptyset},\boldsymbol{u}(\phi(\boldsymbol{u}))=1$和 $W_i^{\emptyset},\boldsymbol{u}(l)=0$，对于任意 $l\in[\![L-1]\!]\setminus\{\phi(\boldsymbol{u})\}$均成立。考虑内部节点 R_i。假设每个子节点 $R_j\in\mathsf{C}_i$的 W-值和 M-值都有定义。接下来，我们为每个元组 $\boldsymbol{u}\in R_i$和每个子节点 $R_j\in\mathsf{C}_i$计算 W-值。具体而言，我们按照节点的递减顺序计算 C_i 中的 $W_i^{j},\boldsymbol{u}$。回顾一下，对于最大的节点 $R_{j^*}\in\mathsf{C}_i$，$W_{i\,\boldsymbol{u}}^{j^*}$、$(0)=1$和 $W_{i\,\boldsymbol{u}}^{j^*}$、$(\ell)=0$对于 $\ell\in[L-1]$成立。接下来，假设已计算出 $W\mathsf{next}(j)_{i,\boldsymbol{u}}(\cdot)$，我们计算$W_i^{j},\boldsymbol{u}(\cdot)$。针对分数 ℓ，我们区分以下两种情况（回顾一下，$\ell\in[\![L-1]\!]$）：

- **情形 1：** $\ell<\phi(\boldsymbol{u})$。在这种情况下，$W_i^{j},\boldsymbol{u}(\ell)=0$因为任何包含 $\boldsymbol{u}$的连接结果，其分数都至少为 $\phi(\boldsymbol{u})$；
- 情形 2：$\phi(\boldsymbol{u})\le\ell<L$。首先，以任意子节点$R_j'\in\mathsf{C}_i$（其位置不早于 R_j）为根的每个子树 $\mathrm{T}j'$ 中，能够与 $\boldsymbol{u}$连接的连接结果的笛卡尔积，本质上就是 T_i^{j} 中能够与 $\boldsymbol{u}$连接的连接结果。我们还需要考虑分数——所有可能的组合 $\ell_j'\in[\![L-1]\!]$：$R_j'\in\mathsf{C}_i,R_j\preceq R_j'$，其中 $\sum_{R_j'}\in\mathsf{C}_i\,\ell_{j'}=\ell-\phi(\boldsymbol{u})$。每种组合都会为 W_i^{j}贡献 $\prod_{R_j'}\in\mathsf{C}_i:R_j\preceq R_j'\ M_{j'\boldsymbol{u}}[\mathsf{key}(j')](\ell_j')$，$\boldsymbol{u}(\ell)$。然而，这类组合的数量高达 $O(L^k)$，这意味着即使只计算一个W-值，也需要 $O(\log^k N)$ 时间。一个关键的观察是，M-值在 R_j' 上与 $R_j\prec R_j'$ 相乘，实际上已由 $W\mathsf{next}(j)_{i,\boldsymbol{u}}$ 捕获，其中 $R\,\mathsf{next}(j)$ 是 C_i 中紧接在 R_j 后面的节点。综合这些观察，我们得到：

$$
\begin{aligned}
W_{i,\boldsymbol{u}}^{j}(\ell) &= \sum_{\substack{\langle \ell_{j'}\in[\![L-1]\!]:R_{j'}\in C_i,R_j\preceq R_{j'}\rangle:\ R_{j'}\in C_i:R_j\preceq R_{j'}\\ \ell-\phi(\boldsymbol{u})=\sum_{R_{j'}\in C_i}\ell_{j'}}}\ \prod M_{j',\boldsymbol{u}[\mathsf{key}(j')]}(\ell_{j'})\\
&= \sum_{\substack{(\ell_1,\ell_2)\in[\![\ell-\phi(\boldsymbol{u})]\!]:\\ \ell_1+\ell_2=\ell-\phi(\boldsymbol{u})}} M_{j,\boldsymbol{u}[\mathsf{key}(j)]}(\ell_1)\cdot \sum_{\substack{\langle \ell_{j'}\in[\![L-1]\!]:R_{j'}\in C_i,R_j\prec R_{j'}\rangle:\ R_{j'}\in C_i:R_j\prec R_{j'}\\ \ell_2=\sum_{R_j\in C_i}\ell_j}}\ \prod M_{j',\boldsymbol{u}[\mathsf{key}(j')]}(\ell_{j'})\\
&= \sum_{\substack{(\ell_1,\ell_2)\in[\![\ell-\phi(\boldsymbol{u})]\!]:\\ \ell_1+\ell_2=\ell-\phi(\boldsymbol{u})}} M_{j,\boldsymbol{u}[\mathsf{key}(j)]}(\ell_1)\cdot W_{i,\boldsymbol{u}}^{\mathsf{next}(j)}(\ell_2). \qquad (8)
\end{aligned}
$$

因此，$W_i^{j},\boldsymbol{u}(\ell)$ 可通过（8）计算得出。此时，子节点 R_j 的 M-值以及后续节点 $R_{\mathsf{next}(j)}$ 的 W-值均已计算完成。注意，$W_i^{\emptyset},\boldsymbol{u}(\ell)$ 可以

类似地通过（8）计算得出：

$$W^{\emptyset}_{i,\boldsymbol{u}}(\ell) = \sum_{\substack{(\ell_1,\ell_2)\in[\![\ell-\phi(\boldsymbol{u})]\!]:\\ \ell_1+\ell_2=\ell-\phi(\boldsymbol{u})}} M_{j^\circ,\boldsymbol{u}[\mathsf{key}(j^\circ)]}(\ell_1)\cdot W^{\mathsf{next}(j^\circ)}_{i,\boldsymbol{u}}(\ell_2),$$

其中 R_{j° 是 C_i 中的第一个节点。

为加快这一过程，我们以一个节点 $R_i \in Q$、一个元组 $\boldsymbol{u} \in R_i$ 和一个子节点 $R_j \in \mathsf{C}_i$ 为例。现在，我们需要计算 $W^{\emptyset}_{i,\ \boldsymbol{u}}(\cdot)$ 对应的 L 个不同分数。$W^{j}_{i,\ \boldsymbol{u}}(\cdot)$、$M_{j,\ \mathsf{key}(j)}(\cdot)$ 与 $W_{\text{下一个}(j)i,\ \boldsymbol{u}}(\cdot)$ 之间存在卷积结构，因此我们使用快速傅里叶变换同时计算这些 L 分数。正如我们将在预处理步骤的分析中所示，这一观察将预处理时间从 $O(NL^2)$ 降至 $O(NL\log L)$（引理 C.2）。

计算完所有W值后，对于每个内部节点 $R_i \in Q$及其每个子节点 $R_j \in \mathsf{C}_i$、每个值 $\boldsymbol{v} \in R_i[\mathsf{key}(i)]$，和每个分数 $\ell \in [\![L-1]\!]$，我们按照预先确定的顺序，在 $W^{j}_{i,\ \boldsymbol{u}}(\ell)\colon \boldsymbol{u} \in R_i \ltimes \boldsymbol{v}$ 上构建一个前缀和数组。

最后，我们计算每个分数 $\ell \in [\![L-1]\!]$ 对应的桶大小 $|\mathsf{B}_\ell|$。根据定义，对于根节点 r，它可以改写为 $|\mathsf{B}^\ell| = \sum_{\boldsymbol{u}\in R_r} W^{\emptyset}_{r,\ \boldsymbol{u}}(\ell)$。

算法 6：预处理(Q)

输入： 连接树 T，关系 $\{R_i\}_{R_i\in\mathcal{T}}$。

对于 节点 $R_i \in \mathcal{T}$，按自底向上顺序**执行**
  如果 R_i 是叶节点，**则**
    对于元组 $\boldsymbol{u} \in R_i$ **执行**
      设置 $W^{\emptyset}_{i,\ \boldsymbol{u}}(\phi(\boldsymbol{u})) \leftarrow 1$ 和 $W^{\emptyset}_{i,\ \boldsymbol{u}}(\ell) \leftarrow 0$ 对于 $\ell \neq \phi(\boldsymbol{u})$;
      $M_{i,\boldsymbol{u}[\mathsf{key}(i)]} \leftarrow M_{i,\boldsymbol{u}[\mathsf{key}(i)]} + W^{\emptyset}_{i,\boldsymbol{u}}$;　　// 逐元素向量加法
  否则// R_i 是一个内部节点
    对每个元组 $\boldsymbol{u} \in R_i$ **执行**
      令 R_{j^*} 为 C_i 中的最后一个子节点;
      设置 $W^{\mathsf{next}(j^*)}_{i,\boldsymbol{u}}(0) \leftarrow 1$ 和 $W^{\text{下一个}(j^*)}_{i,\boldsymbol{u}}(\ell) \leftarrow 0$ 对于 $\ell > 0$;
      对于每个 子节点 $R_j \in C_i$ 按降序 **执行**
        令 $R_{\mathsf{next}(j)}$ 为紧跟在 R_j 之后的子节点;
        对于每个 分数 $\ell \in [\![L-1]\!]$ **执行**
          如果 $\ell < \phi(\boldsymbol{u})$ **则**
            $W^{j}_{i,\boldsymbol{u}}(\ell) \leftarrow 0$;
          else
            $W^{j}_{i,\ \boldsymbol{u}}(\ell) \leftarrow \sum_{k=0}^{\ell-\phi(\boldsymbol{u})} M_{j,\ \boldsymbol{u}[\mathsf{key}(j)]}(k)\cdot W_{\text{下一个}(j)\,i,\ \boldsymbol{u}}(\ell-\phi(\boldsymbol{u})-k)$;
      $M_{i,\boldsymbol{u}[\mathsf{key}(i)]} \leftarrow M_{i,\boldsymbol{u}[\mathsf{key}(i)]} + W^{\emptyset}_{i,\boldsymbol{u}}$;　　// 逐元素向量加法
    遍历 子节点 $R_j \in C_i$ **执行**
      对于每个 $\boldsymbol{v} \in R_i[\text{键}(i)]$ 且 $\ell \in [\![L-1]\!]$，**执行**
        在 $W^{j}_{i,\ \boldsymbol{u}}(\ell)\colon \boldsymbol{u} \in R_i \ltimes \boldsymbol{v}$; 上构建前缀和数组

C.3 优化算法的分析

首先，我们证明算法 4 的正确性。具体而言，我们证明，对于算法 3 中的任意一次 DirectAccess (S_ℓ， j) 调用，算法 4 都会按照每个桶 B_ℓ 中元组的某个固定全序，返回 B_ℓ 中第 τ 个元组。回顾第 2 节，只要元组的顺序固定，其具体顺序并不重要。由于算法 4 是确定性的，只需证明：对于算法 3 中任意两次不同的 DirectAccess ($S_{\hat{\ell}}$， $\hat{\tau}$) 和 DirectAccess ($S_{\bar{\ell}}$， $\bar{\tau}$) 调用，算法 4 返回的两个元组 $\hat{\boldsymbol{w}}$ 和 $\bar{\boldsymbol{w}}$ 均不同，且来自 Join(Q)。

形式化地，我们使用以下引理证明算法 4 的正确性。

引理 C.1。 设 *DirectAccess* ($S_{\hat{\ell}}, \hat{\tau}$) 和 *DirectAccess* ($S_{\bar{\ell}}, \bar{\tau}$) 是算法 *3* 中两个不同的调用。设 $\hat{\boldsymbol{w}} \in$ 连接(Q) 为算法 *4* 在输入 ($S_{\hat{\ell}}, \hat{\tau}$)时返回的元组，$\bar{\boldsymbol{w}} \in$ 连接(Q) 为在输入 ($S_{\bar{\ell}}, \bar{\tau}$)时返回的元组。那么 $\hat{\boldsymbol{w}} \neq \bar{\boldsymbol{w}}$。

证明。注意，由于预处理的正确性，所有W和值 M都已正确计算。

我们首先假设 $\hat{\ell} \neq \bar{\ell}$。我们证明 $\hat{\boldsymbol{w}} \in \mathrm{B}_{\hat{\ell}}$和 $\bar{\boldsymbol{w}} \in \mathrm{B}_{\bar{\ell}}$。考虑算法 3 中任意一次 DirectAccess ($S_{\ell'}, \tau'$) 调用。我们断言，算法 4 总会返回一个属于 $\mathrm{B}_{\ell'}$ 的连接结果。我们对连接树 T 的高度使用强归纳法证明这一点。如果连接树只有一个叶节点 R_i（高度为 1），那么在第 1 行中，算法只考虑分数为 ℓ' 的 R_i 元组，因此返回的元组属于 $\mathrm{B}_{\ell'}$。假设对于所有高度至多为 h 的连接树，该断言都成立。设 R_i 是一棵高度为 $h+1$ 的连接树的根节点。在第 4 行中，算法考虑满足 $\ell_1 + \ell_2 + \phi(\boldsymbol{u}) = \ell'$ 的数对 (ℓ_1, ℓ_2)。根据归纳假设，我们有 $\phi(\boldsymbol{u}_1) = \ell_1$ 和 $\phi(\boldsymbol{u}_2) = \ell_2$，因此 $\phi(\boldsymbol{u}_1 \bowtie \boldsymbol{u} \bowtie \boldsymbol{u}_2) = \ell'$。由此证明了该断言。

将这一论证应用于直接访问 ($S_{\hat{\ell}}, \hat{\tau}$) 和直接访问 ($S_{\bar{\ell}}, \bar{\tau}$)，我们得到 $\hat{\boldsymbol{w}} \in \mathrm{B}_{\hat{\ell}}$和$\bar{\boldsymbol{w}} \in \mathrm{B}_{\bar{\ell}}$。由于 $\hat{\ell} \neq \bar{\ell}$，我们有 $\mathrm{B}_{\hat{\ell}} \cap \mathrm{B}_{\bar{\ell}} = \emptyset$，因此 $\hat{\boldsymbol{w}} \neq \bar{\boldsymbol{w}}$。

现在假设 $\hat{\ell} = \bar{\ell} = \ell^*$ 和 $\hat{\tau} \neq \bar{\tau}$。不失一般性，设 $\hat{\tau} > \bar{\tau}$。我们通过对连接树的高度进行强归纳法来证明该命题。

如果连接树只有一个叶节点 R_i（高度为 1），令 $\hat{\boldsymbol{u}}$（分别为 $\bar{\boldsymbol{u}}$）表示在第 1 行中为数对 $(\hat{\ell}, \hat{\tau})$（分别为 $(\bar{\ell}, \bar{\tau})$）找到的最小元组。对于每个 $\boldsymbol{u}' \in R_i$，值 $W_i^j{}_{,\boldsymbol{u}'}(\ell^*)$ 在$\phi(\boldsymbol{u}') = \ell^*$时等于 1，否则等于 0。因此，求和 $\sum_{\boldsymbol{u}' \in R_i : \boldsymbol{u}' \preceq \boldsymbol{u}} W_i^j{}_{,\boldsymbol{u}'}(\ell^*)$ 会随着每个新符合条件的元组而增加 1。由于 $\hat{\tau} \neq \bar{\tau}$，我们得出 $\hat{\boldsymbol{u}} \neq \bar{\boldsymbol{u}}$，因此 $\hat{\boldsymbol{w}} \neq \bar{\boldsymbol{w}}$。

假设该断言对所有高度至多为 h的连接树均成立。设 R_i 是高度为 $h+1$的连接树的根节点。如果第 1 行中$\hat{\boldsymbol{u}} \neq \bar{\boldsymbol{u}}$，那么显然$\hat{\boldsymbol{w}} \neq \bar{\boldsymbol{w}}$。因此，我们假设$\hat{\boldsymbol{u}} = \bar{\boldsymbol{u}} = \boldsymbol{u}$。对于两个实例，求和$\sum_{\boldsymbol{u}' \in R_i : \boldsymbol{u}' \preceq \boldsymbol{u}} W_i^j{}_{,\boldsymbol{u}'}(\ell^*)$相同，因此在第 2 行中，$\hat{\tau}$和 $\bar{\tau}$减少的量相同。稍微滥用记号，我们再次将更新后的值记为$\hat{\tau}$和 $\bar{\tau}$。设$(\hat{\ell}_1, \hat{\ell}_2)$和$(\bar{\ell}_1, \bar{\ell}_2)$分别是针对输入$(\hat{\ell}, \hat{\tau})$和$(\bar{\ell}, \bar{\tau})$在第 5 行中找到的最小数对。如果$\hat{\ell}_1 \neq \bar{\ell}_1$或$\hat{\ell}_2 \neq \bar{\ell}_2$，则根据证明的第一部分，我们立即得到$\hat{\boldsymbol{w}} \neq \bar{\boldsymbol{w}}$。因此，我们假设$\hat{\ell}_1 = \bar{\ell}_1 = \ell_1$和$\hat{\ell}_2 = \bar{\ell}_2 = \ell_2$。对于两个实例，求和$\sum_{(\ell_1', \ell_2') \in \Phi : (\ell_1', \ell_2') \prec (\ell_1, \ell_2)} M_{j, \boldsymbol{u}[\mathrm{key}(j)]}(\ell_1') \cdot W\mathrm{next}(j)_{i,\boldsymbol{u}}(\ell_2')$相同，因此在第 6 行中，$\hat{\tau}$ 和 $\bar{\tau}$ 再次减少的量相同。

现在我们来到算法的最后阶段，此时最小元组与分数组对相同，但 $\hat{\tau} > \bar{\tau}$。我们按照第 7 行和第 8 行计算 $\hat{\tau}_1$、 $\hat{\tau}_2$ 以及 $\bar{\tau}_1, \bar{\tau}_2$。第 7 行中的分母 $W\mathrm{next}(j)_{i,\boldsymbol{u}}(\ell_2)$ 在两次执行中相同。如果 $\hat{\tau}_1 \neq \bar{\tau}_1$，那么根据归纳假设 $\hat{\boldsymbol{u}}_1 \neq \bar{\boldsymbol{u}}_1$，这意味着 $\hat{\boldsymbol{w}} \neq \bar{\boldsymbol{w}}$。否则，假设 $\hat{\tau}_1 = \bar{\tau}_1 = \tau_1$。有两种情况。如果 $\hat{\tau}$ 和 $\bar{\tau}$都不能被 $W\mathrm{next}(j)_{i,\boldsymbol{u}}(\ell_2)$ 整除，那么 $\hat{\tau}_2 = ((\hat{\tau} - 1) \bmod W\mathrm{next}(j)_{i,\boldsymbol{u}}(\ell_2)) + 1 = \hat{\tau} \bmod W\mathrm{next}(j)_{i,\boldsymbol{u}}(\ell_2) > 0$，且 $\bar{\tau}_2 = ((\bar{\tau} - 1) \bmod W\mathrm{next}(j)_{i,\boldsymbol{u}}(\ell_2)) + 1 = \bar{\tau} \bmod W\mathrm{next}(j)_{i,\boldsymbol{u}}(\ell_2) > 0$。由于 $\hat{\tau} > \bar{\tau}$，可知 $\hat{\tau}_2 > \bar{\tau}_2$，再根据归纳假设 $\hat{\boldsymbol{u}}_2 \neq \bar{\boldsymbol{u}}_2$，再次推出 $\hat{\boldsymbol{w}} \neq \bar{\boldsymbol{w}}$。第二种情况是，$\hat{\tau}$能被 $W\mathrm{next}(j)_{i,\boldsymbol{u}}(\ell_2)$ 整除，但 $\bar{\tau}$不能。于是 $\hat{\tau}_2 = ((\hat{\tau} - 1) \bmod W\mathrm{next}(j)_{i,\boldsymbol{u}}(\ell_2)) + 1 = W\mathrm{next}(j)_{i,\boldsymbol{u}}(\ell_2)$，而

$\bar{\tau}_2 = ((\bar{\tau}-1) \bmod W_{\text{next}(j)i,\boldsymbol{u}}(\ell_2)) + 1 = \bar{\tau} \bmod W_{\text{next}(j)i,\boldsymbol{u}}(\ell_2) < W_{\text{next}(j)i,\boldsymbol{u}}(\ell_2)$。因此 $\hat{\tau}_2 > \bar{\tau}_2$，根据归纳假设 $\hat{\boldsymbol{u}}_2 \neq \bar{\boldsymbol{u}}_2$，由此可得 $\hat{\boldsymbol{w}} \neq \bar{\boldsymbol{w}}$。□

N接下来，我们界定索引的预处理时间、空间和查询时间。

Lemma C.2. 预处理步骤需要 $O(N\log N\log\log N)$ 时间，以及 $O(N\log N)$ 空间。

引理 C.2 的证明。对于任意关系 R_i，以及每个元组 $\boldsymbol{u} \in R_i$，我们存储 $O(|C_i|\cdot L) = O(L)$ 个不同的 W值。因此，存储所有 W值总共需要 $O(NL)$ 的空间。此外，对于每个 $\boldsymbol{v} \in R_i$[键（i），我们存储 $O(L)$ 个不同的 M值。因此，存储这些 M值总共需要 $O(NL)$ 的空间。于是，所使用的总体空间为 $O(NL) = O(N\log N)$。

为了计算每个W值，我们使用公式（5）。根据（5）计算每个 W值，需要对 $O(L)$ 个不同项进行求和，因此，朴素地计算所有 W值的总体运行时间为 $O(NL^2)$。为了加快这一过程，我们以固定节点 $R_i \in Q$、元组 $\boldsymbol{u} \in R_i$ 和子节点 $R_j \in \mathsf{C}_i$ 为例。现在，我们需要为 L 个不同的分数计算 $W_i^j{}_{,\boldsymbol{u}}(\cdot)$。在公式（5）中，$W_i^j{}_{,\boldsymbol{u}}(\cdot)$、$M_{j,\,\text{key}(j)}(\cdot)$ 与 $W_{\text{next}(j)i,\boldsymbol{u}}(\cdot)$ 之间存在卷积结构，因此我们可以应用快速傅里叶变换来同时计算这些 L 分数。这可以将$O(L^2)$计算改进为 $O(L\log L)$。更正式地说，考虑两个次数为 L 的多项式相乘：

$$p(x) = M_{j,\boldsymbol{u}[\text{key}(j)]}(L-1)x^{L-1} + M_{j,\boldsymbol{u}[\text{key}(j)]}(L-2)x^{L-2} + \cdots + M_{j,\boldsymbol{u}[\text{key}(j)]}(0)x^0,$$

and,

$$q(x) = W_{i,\boldsymbol{u}}^{\text{next}(j)}(L-1)x^{L-1} + W_{i,\boldsymbol{u}}^{\text{next}(j)}(L-2)x^{L-2} + \cdots + W_{i,\boldsymbol{u}}^{\text{next}(j)}(0)x^0.$$

根据公式 (5)，注意，对于所有 $\ell \in [\![L-1]\!]$，值$W_i^j{}_{,\boldsymbol{u}}(\ell)$ 等于多项式 $p(x)\cdot q(x)$ 中 $x^{\ell-\phi(u)}$ 的系数。利用快速傅里叶变换，我们可以在 $O(L\log L)$ 时间内计算出次数为 $2L$ 的多项式 $p(x)\cdot q(x)$ 的所有系数。因此，对于固定的节点 $R_i \in Q$、元组$\boldsymbol{u} \in R_i$和子节点 $R_j \in \mathsf{C}_i$，我们可以在 $O(L\log L)$ 时间内找到所有 L 个不同的 $W_i^j{}_{,\boldsymbol{u}}(\cdot)$ 值。综合以上结果，计算所有W值需要 $O(NL\log L) = O(N\log N\log\log N)$。得到 W 值后，利用公式 (4) 在总计 $O(NL)$ 时间内计算 M 值便很直接。因此，预处理步骤的总体运行时间为 $O(N\log N\log\log N)$。

□

引理 C.3。 算法 *4* 的运行时间为 $O(\log N)$。

引理 C.3 的证明。我们将逐步分析其开销。第 1 行对 N个元组在前缀和树上执行二分搜索，耗时 $O(\log N)$。第 3 行同样可以通过前缀和树高效完成，耗时 $O(\log N)$，因为任意前缀和都可以分解为该树中的 $O(\log N)$个规范节点。第 2 行耗时 $O(1)$。第 5–6 行耗时 $O(L) = O(\log N)$，因为最多有 L 对位于 Φ 中，需要对其进行探索。第 7 行耗时 $O(1)$。因此，递归之前的所有行总共耗时 $O(\log N)$。由于最多有 k^2种不同的 (i, j) 组合，并且对于每种组合 (i, j) 只调用一次递归，因此递归总次数为$O(k^2)$。由于 k是常数，总运行时间为 $O(\log N)$。□

最后，我们准备好证明定理 3.3。

定理 3.3 的证明。预处理步骤的开销可直接由引理 C.2 得出。下面，我们着重讨论查询回答时间。在回答子集采样查询时，每当需要从 B_ℓ 中检索某个连接结果，我们区分以下两种情况。如果 $\ell = L$，我们会计算 $\mathsf{B}_{\geq L}$ 中的全部连接结果（如果这些连接结果之前尚未计算），以支持直接访问。否则，我们会相应地调用 **RecursiveAccess** 过程。对于第一种情况，预期开销为 $O(1)$，因为 $\mathsf{B}_{\geq L}$ 中至多有 $O(N^{\rho^*})$ 个连接结果，需要花费

$O(N^{\rho^*})$ 的时间来物化 [5],，且从 $\mathrm{B}_{\geq L}$ 中检索到至少一个连接结果的概率至多为 $1-(1-p_L^+)^{|\mathcal{B}_{\geq L}|} \leq 1-(1-\frac{1}{N^{2\rho^*}})^{N^{\rho^*}} \leq \frac{1}{N^{\rho^*}}$，因为 $|\mathrm{B}_{\geq L}| = N^{\rho^*}$，且 $p_L^+ \leq \frac{1}{2^L} \leq \frac{1}{N^{2\rho^*}}$。根据引理 2.2 和引理 C.3，预期查询回答时间为 $O(1+\mu_\Psi \log N)$。□

D 第 4 节中缺失的材料

算法 7 的分析。正确性与算法 4 相同，因为我们精确模拟了算法 4 的所有调用的执行。由定理 3.3 得到的数据统计量可在 $O(N\log N\log\log N)$ 时间内计算。所有 X-数组均可在 $O(N\log N)$ 时间内计算，因为对于每个 $\boldsymbol{u} \in R_i$，都存在唯一的 $\boldsymbol{v} \in R_i[\text{键}(i)]$ 使得 $\boldsymbol{u} \in R_i \ltimes \boldsymbol{v}$。所有 Y- 数组均可在 $O(N\log^2 N)$ 时间内计算，因为共有 $O(N\log N)$ 个数组，且每个数组的大小为 $O(\log N)$。利用预先计算的 M 和 W 统计量，每个 Y 数组中的每个值都可在 $O(1)$ 时间内计算。

固定一个表 R_i 及其在 T 中的一个子节点 R_j。由于 $|\mathrm{P}_{i,j}| = O(1+\mu_\Psi)$，基数排序的期望运行时间为 $O(N+\mu_\Psi)$。为了计算 $\mathrm{P}_{i,j,\boldsymbol{v},\ell}$ 中每个五元组对应的最小元组 $\boldsymbol{u}$，我们遍历 $\mathrm{P}_{i,j,\boldsymbol{v},\ell}$ 中的所有五元组，并在最坏情况下遍历 $X_{i,j,\boldsymbol{v},\ell}$ 一次。由于对于每个 $\boldsymbol{u} \in R_i$，都存在唯一的 $\boldsymbol{v}$ 满足 $\boldsymbol{u} \in R_i \ltimes \boldsymbol{v}$，且 $\mathrm{P}_{i,j}$ 中的每个五元组都属于唯一的分组 $\mathrm{P}_{i,j,\boldsymbol{v},\ell}$，因此有 $\sum_\ell (|X_{i,j,\boldsymbol{v},\ell}| + |\mathrm{P}_{i,j,\boldsymbol{v},\ell}|) = O(N\log N + \mu_\Psi)$。类似地，计算所有数对 (ℓ_1,ℓ_2) 需要对每个 $\mathrm{P}_{i,j,\boldsymbol{u},\ell}$ 中的所有五元组以及所有 Y 表进行一次遍历。由于有 $O(N\log N)$ 个表，Y 每个表的大小为 $O(\log N)$，总成本为 $O(N\log^2 N + \mu_\Psi)$ 期望时间。T 中的节点数为 $k = O(1)$，且每个节点有 $O(1)$ 个子节点，因此，单次算法的总期望运行时间为 $O(N\log^2 N + \mu_\Psi)$。

E 关于其他函数的讨论

接下来，我们讨论如何扩展到其他聚合函数。对于我们的算法，可以将结果扩展到支持 MIN、MAX 和 SUM。我们将展示如何把第 3.1 节中的高层思路应用于这些函数。

MIN 和MAX。不失一般性，我们考虑函数 MIN。假设输入连接 Q 可使用一个 DirectAccess 预言机（按照某种固定顺序），该预言机接收一个整数 $i \in [|\mathrm{Join}(Q)|]$，并返回 $\mathrm{Join}(Q)$ 中的第 i 个元素。与之前类似，顺序可以任意指定，但在多次调用该预言机时必须保持一致。令 $L = \lceil 2\rho^* \log N \rceil$。划分连接结果的第一步与之前完全相同。第二步也同样是对所有子实例应用子集采样。令 $\boldsymbol{j} = (j_1, j_2, \ldots, j_k)$。子实例 $Q_{\boldsymbol{j}}$ 中的每个连接结果被抽样的概率均为 $\min_{i\in[k]} 2^{-j_i} = 2^{-\max_{i\in[k]} j_i}$。我们先指出这种划分中每个子实例的以下性质：

引理 E.1（轻型或近似均匀子实例）。令 $\boldsymbol{j} = (j_1, j_2, \ldots, j_k)$。若最大值 $i \in [k] j_i \geq L$，则实例 $\Psi_{\boldsymbol{j}}$ 是轻量的；否则，它是2-均匀分布的。

直观而言，若 $\max_{i\in[k]} j_i \geq L$，则每个连接结果的概率至多为 $\frac{1}{|\text{连接}(Q)|}$，因此根据定义，该子实例是轻量的。否则，$\max_{i\in[k]} j_i < L$。此时，最大概率与最小概率之比至多为 2，因此根据定义，该子实例是 2-均匀分布的。因此，

- **预处理阶段：**我们按权重划分输入元组以及连接实例 Q。对于每个满足 $\boldsymbol{j} \in [\![L]\!]^k$ 的子实例 $Q_{\boldsymbol{j}}$，我们构建直接访问预言机并计算连接大小。
- **查询阶段：**我们使用输入 $\{Q_{\boldsymbol{j}},\ p : \boldsymbol{j} \in [\![L]\!]^k\}$ 且满足 $p_{\boldsymbol{j}}^+ = 2^{-\text{最大值}_{i\in[k]} j_i}$ 来调用算法 3。每当从 $Q_{\boldsymbol{j}}$ 中抽取一个连接结果时，我们就使用直接访问预言机来获取该结果。

对于优化索引，对每个 $\boldsymbol{u}$ 属于 R_i 的对象，我们定义分数 $\phi(\boldsymbol{u}) = \lfloor -\log \mathrm{p}_i(\boldsymbol{u}) \rfloor$。对于连接结果 $\boldsymbol{u} \in \mathrm{Join}(Q)$，其分数为 $\bar{\phi}(\boldsymbol{u}) = \text{最小值}_{i=1}^k \phi(\boldsymbol{u}[\text{模式}(R_i)])$。桶 $\mathrm{B}_\ell = \{\boldsymbol{u} \in \mathrm{Join}(Q) \mid \bar{\phi}(\boldsymbol{u}) = \ell\}$ 的定义方式与乘积函数类似。注意，对于任意 $\boldsymbol{u} \in \mathrm{B}_\ell, 2^{-\ell-1} \leq \mathrm{p}(\boldsymbol{u}) \leq 2^{-\ell}$，因此每个桶中的最大概率与最小概率之比

算法 7：BatchRecursiveAccess$(i, j, \mathrm{P}_{i,j})$

$\mathcal{U} \leftarrow \emptyset$;
使用基数排序对 $\mathrm{P}_{i,j}$ 进行排序；
进一步按 $(i, j, \boldsymbol{v}, \ell)$ 对 $\mathrm{P}_{i,j}$ 分组，创建分组 $\mathrm{P}_{i,j,\boldsymbol{v},\ell}$，其排序顺序依据 $\mathrm{P}_{i,j}$;
对于 $\ell \in [\![L-1]\!]$，**执行**
  对于每个$(i, j, \boldsymbol{v}, \ell, \tau) \in \mathcal{P}_{i,j,\boldsymbol{v},\ell}$按排序顺序**执行**
    $\boldsymbol{u} \leftarrow X_i$中的第一个元组$_{,j,\boldsymbol{v},\ell}$;
    当$X_{i,j,\boldsymbol{v},\ell}(\boldsymbol{u}) < \tau$**执行** $\boldsymbol{u} \leftarrow X_{i,j,\boldsymbol{v},\ell}$ 中的下一个元组;
    如果 R_i是叶节点**则**将 $\boldsymbol{u}$作为 $\vec{\boldsymbol{u}}(i, j, \boldsymbol{v}, \ell, \tau)$添加到$\mathcal{U}$;
    $\mathrm{P}_{i,j} \leftarrow (\mathrm{P}_{i,j} - \{(i, j, \boldsymbol{v}, \ell, \tau)\}) \cup \{(i, j, \boldsymbol{u}, \ell, \tau - X_{i,j,\boldsymbol{v},\ell}(\mathrm{prev}(\boldsymbol{u})))\}$;
如果 R_i 是叶节点**则返回**$\mathcal{U}$;
使用基数排序对 $\mathrm{P}_{i,\ j}$ 进行排序；
按 $(i,\ \ j,\ \ \boldsymbol{v},\ \ \ell)$ 对 $\mathrm{P}_{i,\ j}$进一步分组，创建分组 $\mathrm{P}_{i,\ j,\ \boldsymbol{v},\ \ell}$，依据按 $\mathrm{P}_{i,\ j}$ 的排序顺序。；
$\mathcal{P}_{j,\emptyset} \leftarrow \emptyset, \mathrm{P}_{i,\mathrm{next}(j)} \leftarrow \emptyset$;
对于 $\ell \in [\![L-1]\!]$，**执行**
  对于 每个 $(i,\ \ j,\ \ \boldsymbol{u},\ \ \ell,\ \ \tau) \in \mathcal{P}_{i,\ j,\ \boldsymbol{u},\ \ell}$按排序顺序执行**执行**
    $(\ell_1,\ \ \ell_2) \leftarrow$ 在 $Y_{i,\ j,\ \boldsymbol{u},\ \ell}$ 中的第一个数对;
    当$Y_{i,\ j,\ \boldsymbol{v},\ \ell}((\ell_1,\ \ \ell_2)) < \tau$**时**，$\ell_1,\ \ \ell_2$ 是 $Y_{i,\ j,\ \boldsymbol{u},\ \ell}$ 中的下一个数对;
    根据算法 4 第 7 行，基于 $\ell_1,\ \ \ell_2$ 计算 $\tau_1,\ \ \tau_2$；
    $\mathcal{P}_{j,\emptyset} \leftarrow \mathrm{P}_{j,\ \emptyset} \cup \{(j,\ \ \emptyset,\ \ \boldsymbol{u},\ \ \ell_1,\ \ \tau_1)\}$;
    $\mathcal{P}_{i,\mathrm{next}(j)} \leftarrow \mathrm{P}_{i,\ \mathrm{next}(j)} \cup \{(i,\ \ \mathrm{next}(\ j)\ \boldsymbol{u},\ \ \ell_2,\ \ \tau_2)\}$;
$\mathcal{U} \leftarrow \mathcal{U} \cup$ **BatchRecursiveAccess**$(j, \emptyset, \mathcal{P}_{i,\emptyset})$;
$\mathcal{U} \leftarrow \mathcal{U} \cup$ **BatchRecursiveAccess**$(i,\ \ \mathrm{next}(\ j),\ \ \mathcal{P}_{i,\ \mathrm{next}(j)})$;
对于$(i, j, \boldsymbol{u}, \ell, \tau) \in \mathcal{P}_{i,j}$**执行**
  $\vec{\boldsymbol{u}}(i,\ \ j,\ \ \boldsymbol{u},\ \ \ell,\ \ \tau) \leftarrow \vec{\boldsymbol{u}}(\ j,\ \ \emptyset,\ \ \boldsymbol{u},\ \ \ell_1,\ \ \tau_1) \bowtie \boldsymbol{u} \bowtie \vec{\boldsymbol{u}}(i,\ \ \mathrm{next}(\ j)\ \boldsymbol{u},\ \ \ell_2,\ \ \tau_2)$;
  // $\ell_1, \ell_2, \tau_1, \tau_2$ 在上文第 17 行和第 18 行中计算得出
  将 $\vec{\boldsymbol{u}}(i, j, \boldsymbol{u}, \ell, \tau)$ 添加到 U 中;
返回$\mathcal{U}$;

桶 B_ℓ 至多为 2。该索引与我们在第 3.2 节中构建的索引类似。在查询阶段，我们运行一个类似于算法 4 的算法。一个区别是，在第 4 步和第 5 步中，我们尝试的所有数对不再像使用乘积函数时那样是 $L = O(\log\ N)$，而是 $O(L^2) = O(\log^2 N)$。总体而言，定理 3.3 可扩展到最小值（以及最大值）函数，保证不变，但多出一个 $\log N$ 因子。同样，一次性和动态变体的结果也可以扩展，但会额外多出一个 $\log N$ 因子。

求和。类似地，我们的索引框架也可以自然地扩展到 SUM 函数。有趣的是，SUM 可以完全按照与最大值相同的方式处理：对于 SUM 函数，我们定义与最大值函数所使用的完全相同的分数和桶。也就是说，对于每个 $\boldsymbol{u}$ in R_i，我们定义分数$\phi(\boldsymbol{u}) = \lfloor -\log \mathrm{p}_i(\boldsymbol{u}) \rfloor$。对于连接结果 $\boldsymbol{u} \in \mathrm{Join}(Q)$，其分数为 $\bar{\phi}(\boldsymbol{u}) = \max_{i=1}^{k} \phi(\boldsymbol{u}[\mathrm{schema}(R_i)])$。¯对于每个 ℓ，我们定义桶 $\mathrm{B}_\ell = \{\boldsymbol{u} \in \mathrm{Join}(Q) \mid \phi(\boldsymbol{u}) = \ell\}$。注意，对于任意 $\boldsymbol{u} \in \mathrm{B}_\ell$，$2^{-\ell-1} \le \mathrm{p}(\boldsymbol{u}) \le k \cdot 2^{-\ell}$，因此每个桶 B_ℓ中的最大概率与最小概率之比至多为 $2k = O(1)$。该索引与我们在第 3.2 节中构建的索引类似。在查询阶段，我们运行一个类似于算法 4 的算法。一个区别是，在步骤 4 和 5 中

我们尝试的所有数对不是像使用乘积函数时那样的 $L = O(\log N)$，而是 $O(L^2) = O(\log^2 N)$。总体而言，定理 3.3 可以扩展到 SUM 函数，并且除一个 $\log N$ 因子外具有相同的保证。类似地，一次性方法和动态变体的结果也可以扩展，但还需额外增加一个 $\log N$ 因子。

F 定理 5.3 的证明

引理 F.1. 摊销更新时间为 $O(\log^3 N \log\log N)$。

证明。关键观察是，对于每个关系 R_i 及其父节点 R_p，当且仅当 $\tilde{M}_{i,\boldsymbol{v}}(\ell)$ 发生变化时，我们才会更新值 $\tilde{W}^j_{p,\boldsymbol{u}}(\ell')$，其中 $\boldsymbol{u} \in R_p \ltimes \boldsymbol{v}$（注意，对每个 $\boldsymbol{u}$，都存在唯一的 $\boldsymbol{v}$ 使得 $\boldsymbol{u} \in R_p \ltimes \boldsymbol{v}$）。对于任意固定的 $i, \boldsymbol{v}$, 和 ℓ，值 $M_{i,\boldsymbol{v}}(\ell)$ 至多变化 $O(\log N)$ 次。

每当发生更新时，我们都会使用 FFT 算法重新计算每个 $\ell' \in [\![L-1]\!]$ 对应的所有值 $\tilde{W}^j_{p,\boldsymbol{u}}(\ell')$，耗时为 $O(L \log L)$。由于可能的分数有 L 个，对于固定的 i 和 $\boldsymbol{v}$，存在 $O(L)$ 个不同的值 $\tilde{M}_{i,\boldsymbol{v}}(\ell)$。因此，值 $\tilde{W}^j_{p,\boldsymbol{u}}(\ell')$ 总共至多更新 $O(L \log N)$ 次，每次耗时 $O(L \log L)$。

这将产生 $O(\log^3 N \log\log N)$ 的摊销更新时间。

最后，每经过 N 次插入，我们就从头开始重建动态索引，并更新 $N \leftarrow 2N$。此重建步骤保持相同的摊销更新时间界，即 $O(\log^3 N \log\log N)$。□

引理 F.2。 该算法能够正确更新值 W、M 和 M。

证明。不失一般性，假设在第一次调用 Update$(i, \boldsymbol{v}, \ell, \Delta)$ 时，子树 T_i 上的所有值 W、M 和 M 一直得到正确维护；该调用发生在 T 的层级 h 处的节点 R_i。令 Δ 表示 $M_{i,\boldsymbol{v}}(\ell)$ 的正确增量。

我们分两种情况讨论。如果 $M_{i,\boldsymbol{v}}(\ell)$ 不发生变化，那么根据公式（7），R_i 的任何祖先节点都不需要更新值 W。由于 W 保持不变，R_i 的祖先节点上的所有对应 M 值也保持不变。因此，无需进行进一步更新。

否则，假设 $\tilde{M}_{i,\,\boldsymbol{v}}(\ell)$ 发生变化。随后，我们根据公式（7）重新计算所有值 $\tilde{W}^j_{p,\,\boldsymbol{u}}(\ell')$（算法 5，第 7 行）。如果某个值 $\tilde{W}^j_{p,\,\boldsymbol{u}}(\ell')$ 随 $j \neq \emptyset$ 增大，那么祖先节点 R_i 中的其他 $\tilde{W}$ 或 M 值都无需更新。如果某个值 $\tilde{W}^{\emptyset}_{p,\,\boldsymbol{u}}(\ell')$ 未增大，则同样无需更改其祖先节点中的值。

最后，如果 $\tilde{W}^{\emptyset}_{p,\,\boldsymbol{u}}(\ell')$ 增加了非零量 Δ'，那么根据公式（6），值 $M_{i,\,\boldsymbol{v}}(\ell')$ 必须增加 Δ'。因此，将触发递归调用 **更新**$(p,\ \boldsymbol{u}[\mathrm{key}(p)],\ \ell',\ \Delta')$，并在算法 5 第 2 行更新 $M_{i,\,\boldsymbol{v}}(\ell')$。递归地应用相同的论证，即可证明整个更新过程的正确性。□

引理 F.3. 查询过程的期望时间为 $O(1 + \mu_\Psi \cdot \text{对数 } N)$。

证明。根据定义，存在一个常数 $c^\star$，它仅依赖于 $|\mathrm{T}|$，使得

$$W^j_{i,\boldsymbol{u}}(\ell) \le \tilde{W}^j_{i,\boldsymbol{u}}(\ell) \le c^\star W^j_{i,\boldsymbol{u}}(\ell) \quad \text{and} \quad M_{i,\boldsymbol{u}}(\ell) \le \tilde{M}_{i,\boldsymbol{u}}(\ell) \le c^\star M_{i,\boldsymbol{u}}(\ell),$$

对于所有 i、j、ℓ 和 $\boldsymbol{u}$ 均成立。因此，我们可以模拟算法 3（以及算法 4），并考虑每个桶 B_ℓ 中的元素数量为 $M_{r,\emptyset}(\ell) = O(M_{r,\emptyset}(\ell))$，其中 r 是 T 的根节点。换句话说，我们的索引隐式地包含真实连接结果的一个超集，其中可能包括虚拟元组。由于非虚拟元组的数量是该超集的一个常数比例，我们得到查询过程在 $O(1 + \mu_\Psi \log N)$ 期望时间内运行。事实上，令 Ψ 为考虑真实 $W^j_{i,\boldsymbol{u}}(\ell)$、$M_{i,\boldsymbol{u}}(\ell)$ 的子集采样实例，并令 $\tilde{\Psi}$ 为考虑近似值 $\tilde{W}^j_{i,\boldsymbol{u}}(\ell)$、$\tilde{M}_{i,\boldsymbol{u}}(\ell)$ 的实例。对于 $\ell \in [\![L-1]\!]$，令 κ_ℓ 为来自实例 Ψ 的元组在桶 B_ℓ 中的数量。对于每个桶 B_ℓ，实例 Ψ 包含来自实例

Ψ 同一桶中的所有元组，以及 $\tilde{c} \cdot \kappa_\ell$ 个额外的虚拟元组，其中 $\tilde{c}$ 是取决于|T |的常数。因此，$\mu_{\Psi'} = O(\mu_\Psi)$，结论得证。□

引理 F.4. 查询过程返回一种有效子集采样。

证明。由于$W_i^j(\boldsymbol{u}, \ell) \le \tilde{W}_i^j(\boldsymbol{u}, \ell) \le c^\star W_i^j(\boldsymbol{u}, \ell)$ 和 $M_i(\boldsymbol{u}, \ell) \le \tilde{M}_i(\boldsymbol{u}, \ell) \le c^\star M_i(\boldsymbol{u}, \ell)$，对于所有 i, j, ℓ 和$\boldsymbol{u}$，正确性可直接由算法 3 和引理 2.4 得出，其中 $p_\ell^+ = c \cdot 2^{-\ell}$，对于一个依赖于 T 的˜常数 c，且对于每个 $\ell \in [\![L-1]\!]$，$|S_\ell| = |\mathrm{B}_\ell| = M_r(\emptyset, \ell)$。□