随机算法课程

实验报告

实验五：大数据抽样——Random Sampling over Joins Revisited

姓名：袁野

学号：1190200122

班级：1903102

评分表：（由老师填写）

最终得分：	
对实验题目的理解是否透彻：	
实验步骤是否完整、可信 ：	
代码质量 ：	
实验报告是否规范 ：	
趣味性、难度加分 ：	
特 色：	1
	2
	3

一、实验题目概述

本次实验我选择的论文为《Random Sampling over Joins Revisited》。

在数据库中，对于若干个表的连接结果的随机取样并不是一件容易的事。由于关系表内的数据量很大，并且在关系表连接之后的结果是单个表数据量的次方关系，数量十分庞大，处理出来、储存、并随机取样是一件十分困难的事情，同时由于存在 $sample(R \bowtie S) \neq s\text{ample}(R) \bowtie s\text{ample}(S)$ 的关系，因此我们并不能直接在每一个表上随机采样然后再将其连接的操作。

这篇文献提供了一些算法可以帮助我们以较小的代价实现准确的或近似的均匀采样并进行了性能的分析。

二、对文献内容进行分析

2.1 背景

考虑下面的场景，如果我们想得知多位顾客的购物习惯，以及对生产厂家的喜好等的信息，最直观的办法就是找到顾客购物的关系表，商店所出售货物的关系表，以及货物和供应商的关系表等，再将这些表进行连接，将连接的结果放入训练模型中进行训练。但是实际上，这些关系表在进行连接之后的结果数量会非常庞大，很难将其完全放入训练模型，因此我们可以考虑在这些结果中进行均匀的多次随机抽样，再将抽样结果放入训练模型中进行训练。可以证明的是，这样的抽样结果的训练绝大多数的情况下是可以替代将全部数据放入训练模型进行训练的。这时我们就希望有这样一种抽样方法，使得其可以在可接受的空间复杂度和时间复杂度下进行抽样，且如果最后连接之后的结果数量为 n，那么每一个结果元组被抽到的概率均为 $\frac{1}{n}$。

2.2 前置知识

这篇文献首先介绍了前人的一些两种针对两个表之间连接的抽样算法。

1) Olken’s algorithm for 2-table joins

假设有两张表 R1 和 R2，R1 中有两列分别为 A，B，R2 中同样有两列分别为 B，C。首先对 R1 中的元组做均匀采样，结果为 t_1，然后对 R2 中与 t_1 直接连接的元组中做均匀采样，然后以 $\alpha = \frac{d_B(\pi_B(t_1),R_2)}{M_B(R_2)}$ 的概率接受它。

这样的话抽样结果并接收的概率我们可以计算为

$$Pr(t_1, t_2 \wedge accepted) = Pr(t_1) \times Pr(t_2) \times \alpha$$

那么有

$$\begin{aligned} Pr(t_1, t_2 | accepted) &= \frac{Pr(t_1, t_2 \wedge accepted)}{Pr(accepted)} \\ &= \frac{Pr(t_1, t_2 \wedge accepted)}{\sum_{t_1,t_2|t2 \in t1R2} Pr(t_1, t_2 \wedge accepted)} \\ &= \frac{1}{n} \end{aligned}$$

其中 n 为表关系连接后的元组数量，$d_B(t_x, R_y)$ 为元组 t_x 与关系表 R_y 中可连接的元组的个数，$\pi_B(t_x)$ 为 t_x 向 B 上的投影，$M_B(R_2)$ 为 B 中

在 R_2 中某一元素出现的最大频次。可以通过举例验证上述表达式的结果为连接之后结果中元组个数的倒数。

2) Chaudhuri et al.’s algorithm for 2-table joins

其他描述与上述一致，这个算法的步骤为，首先以与 $d_B(b,R_i)$ 成比例的概率在 R1 中进行抽样，结果为 t_1，然后在 R2 中所有与 t_1 直接连接的元组进行均匀采样。那么 t_1,t_2 即为采样的最终结果。这样的话并不需要拒绝采样结果。我们来计算这样采样的话连接结果中每个元组被抽到的概率

$$\begin{aligned} Pr(t_1,t_2) &= \frac{d_B(t_1,R_2)}{\sum_{i=1}^{|R_1|} d_B(t_i,R_2)} * \frac{1}{d_B(t_1,R_2)} \\ &= \frac{1}{\sum_{i=1}^{|R_1|} d_B(t_i,R_2)} \\ &= \frac{1}{n} \end{aligned}$$

这个计算过程是比较显而易见的。

3) Acharya et al.’s algorithm for multi-way foreign-key joins

这个算法是面向 multi-way foreign-key joins 问题的，即 R_i 中每一个元素的外键是 R_{i+1} 中某一个元素的主键。这样的重多表连接之后的结果中的元组数量即为 R_1 中元组的数量，且其第一列的值互不相同。这个算法的采样过程即为直接对 R_1 中的元组进行均匀采样，那么此时最终的采样结果也就确定了，只要沿着外键不停的往后连接就行了。

2.3 multi-way joins 的采样算法的普适性框架

假设我们现在有 n 张表，每张表都有两列，第一张表为 A、B，第二张表为 B、C，第三张表为 C、D…… 我们要对这n 张表的连接结果进行均匀采样。

首先第一步是我们额外添加一个连接头 t_0，这个元组与 R1 中的每一个元组都可以连接。

然后对于每一个$t_i \in R_i$，我们设权重 $w(t) = |t_i \bowtie R_{i+1} \bowtie \cdots \bowtie R_n|$，$w(R) = \sum_{t\in R} w(t)$，那么就有 $w(R_0) = w(t_0) = |R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n|$。

通常情况下我们并不会将 $w(t)$ 的精确值求出来，那么此时我们可能会计算出一个其上界 $W(t)$，满足

- $\forall t, W(t) \geq w(t)$
- $\forall t \in R_n, W(t) = w(t) = 1$
- $\forall 0 \leq i \leq n-1, \forall t_i \in R_i, W(t_i) \geq W(t_i \rtimes R_{i+1})$

那么在每次采样时，我们就会以 $1 - \frac{W(t_i \rtimes R_{i+1})}{W(t_i)}$ 的概率拒绝这次采样，如果接收，那么接下来以 $\frac{W(t_{i+1})}{W(t_i \rtimes R_{i+1})}$ 的概率对 R_{i+1} 所有与 t_i 直接相连的元素进行采样。

这样的话我们计算每一个结果中的元组被采样到且接收的概率

$$Pr(t \wedge accepted) = \prod_{i=0}^{n-1} \frac{W(t_i \rtimes R_{i+1})}{W(t_i)} \cdot \frac{W(t_{i+1})}{W(t_i \rtimes R_{i+1})} = \frac{1}{W(t_0)}$$

$$\Rightarrow pr(t|accepted) = \frac{Pr(t \wedge accepted)}{Pr(accepted)} = \frac{1/W(t_0)}{|J|/W(t_0)} = \frac{1}{|J|}$$

算法流程为：

Algorithm 1: Sampling over a chain join

Input: $R_1, \cdots R_n, W(t)$ for t_0 and any $t \in R_i, i \in [1, n]$
Output: A tuple sampled from $R_1 \bowtie \cdots \bowtie R_n$ or reject

```
t ← r_0;
S ← (r_0);
for i = 1, ..., n do
    W'(t) ← W(t);
    W(t) ← W(t ⋊ R_i);
    reject with probability 1 − W(t ⋊ R_i)/W'(t) ;
    t ← a random tuple t' ∈ (t ⋊ R_i) with probability
      W(t')/W(t ⋊ R_i);
    add t to S;
end
report S;
```

那么我们如何去估计 $W(t)$ 即为当前问题的重点。这篇文献中提出了三种估计方法，我们将在下边提到。

2.4 权重的估计算法

1) Exact Weight

这种方法是直接根据 $w(t)$ 的定义，利用动态规划的方式去计算所有元组的 $w(t)$ 的值，那么每次采样的话直接对于 R_{i+1} 所有与 t_i 直接相连的元素 t_{i+1}，按照与 $\frac{w(t_{i+1})}{t_i}$ 成正比的概率进行抽样，有上一部分 multi-way joins 的采样算法的普适性框架中的内容可知，此时拒绝率总为 0。这种方法实际上是上述 Chaudhuri et al.’s algorithm 的拓展。

这种方法在求 $w(t_i)$ 时会相对比较慢，但是由于其权重为准确值，因此采样时并不会拒绝采样，所以其采样的速度时相对比较快的。

2) Extended Olken

这种方法并没有对 $w(t)$ 进行直接求得，而是将每一张表所有元组中连向下一张表的元组最大个数与下一张表中任意一个元组的 $W(t_{i+1})$（其实取值是相同的）的乘积，直接赋值给这一张表中所有元素的 $W(t_i)$ 即可。

实际上由于这样的计算方法会导致最终 $W(t_i)$ 和 $w(t_i)$ 差距过大，这样的结果就是拒绝率直线上升。作者还引入了另外一种 AGM Bound。这两种上界计算方式各有优劣，对于出现频次较小的取值我们采用 Olken Bound，否则采用 AGM Bound，那么我们就对于不同的连接采用不同的上界计算方式并相乘，并将最终结果相加即可得到一个相对比较优秀的估值。

这个算法预处理 $W(t_i)$ 的速度很快，而由于其权重误差较大，因此拒绝率会很高，采样所需的时间就会很长。

3) Online Exploration

这种方法实际上是借助了随机游走进行加速 Exact Weight 算法。首先我们进行大量的随机游走，每次等概率的选择 R_{i+1} 所有与 t_i 直接相连的元素 t_{i+1}，并且统计每一个元组被随机游走经过的次数。对于所有随机游走经过次数超过一定阈值的元组，我们用 wander join estimator 进行权重的估计，而对于其他的元组，我们采用动态规划的方式对其权重进行计算。

我们不难发现，这个算法的效率对于随机游走的次数和阈值的选择是十分依赖的，如果这两个参数调整的得当，其计算权重数组的时间效率会优于 Exact Weight，且由于权重的相对误差较小，拒绝率小，其采样的时间效率远优于 Extended Olken。

4) Reverse Sampling

这个算法就是 Acharya et al.’s algorithm for multi-way foreign-key joins 的拓展，其实现方式完全相同。

三、对实验步骤的详细阐述

1、对于 Reverse Sampling 算法，我们生成的关系表满足 multi-way foreign-key joins 的要求，并且单独运行其结果。

2、对于另外三个算法，由于其均符合 multi-way joins 的采样算法的普适性框架，唯一的区别就是每一个元组的权重的计算方式不同，因此我将该框架的采用一个类来实现，其中求权重部分用虚函数定义。这三种采样方式分别继承原框架类，并分别实现权重求值部分即可。

3、我在实验中测试了两个指标，首先是各个元组抽样的均匀性，其次是权重求值时间和采样时间的比较。这两个指标所采用的数据规模是不一样的。

4、检验均匀性时，会生成 3 个表，每个表均包含 20 个元组，然后进行 256000000 抽样，用一个 map 记录每一个结果被抽到次数。

5、检验效率时，会生成 10 个关系表，每个关系表中有 5000 个元组，然后进行 1000000 次抽样，并统计时间。

6、首先我会将所有的数据读到内存中，并用多次双重循环将相邻两个关系表中后一个表中可以连接的元组行号存入前一个表的元组的 nxt 中，nxt 为动态数组。上边的操作是为了构建类似索引的数据结构，从而减少不必要的时间消耗。然后采用各自算法将每一个元组的权重计算出来，并记录其运行时间。接着我在对每一个元组新建一个数组，记录其下一个表中可与其直接连接的元组们权重的前缀和，记录前缀和的目的是为了如果要按照权重等比例进行随机采样时，可以根据生成的随机数，通过二分查找的方式快速寻找到其落到的区间，从而确定随机到的对象。最后便进行多次采样，并记录下所需要的时间。

7、Online Exploration 方法中阈值设置为 13，随机游走次数设置为表内元组的平均个数的 100 倍。

四、实验数据

1. 实验设置

实验环境：

Ubuntu20.04.4 LTS(GNU/Linux 5.10.102.1-microsoft-standard-WSL2 x86_64)

数据：

每一个文件代表一个关系表，每一行对应一个元组，每个元组有三列，最左边为与前一个表的等值连接依据，中间为数据值，其与行号对应，方便对采样的元组进行输出调试，最右侧为与下一张表的等值连接依据。

每个元组最左侧的取值为 [1,10]，且取值为 1，2，3，4，5，6，7，8，9，10 的概率比为 1:2:3:4:5:6:7:8:9:10。每个元组最右侧的取值也为 [1,10]，且取到每个值的概率相等。

2. 实验结果

四种取样方法下每个元组被采样到的次数折线图如下

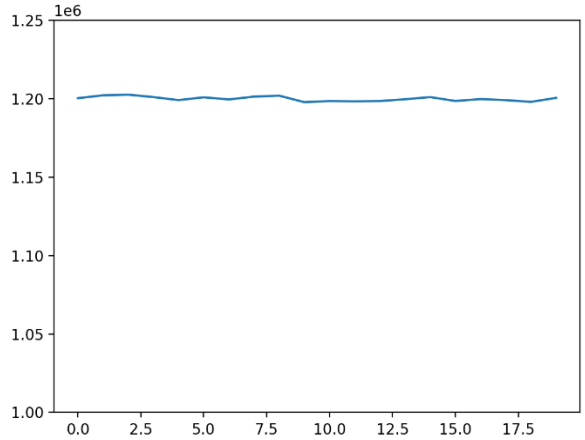


图 1 Reverse Sampling 结果元组与其抽样次数的关系

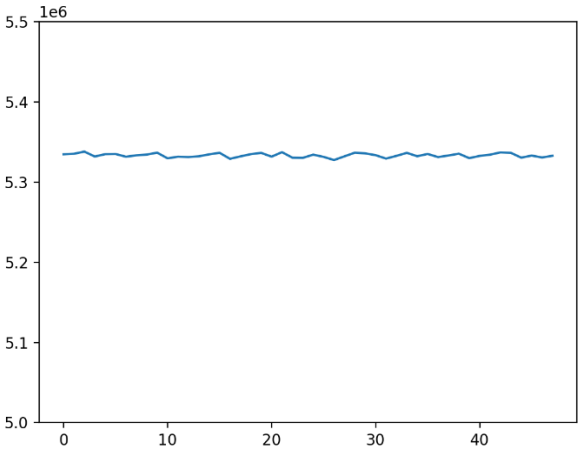


图 2 Exact Weight 结果元组与其抽样次数的关系

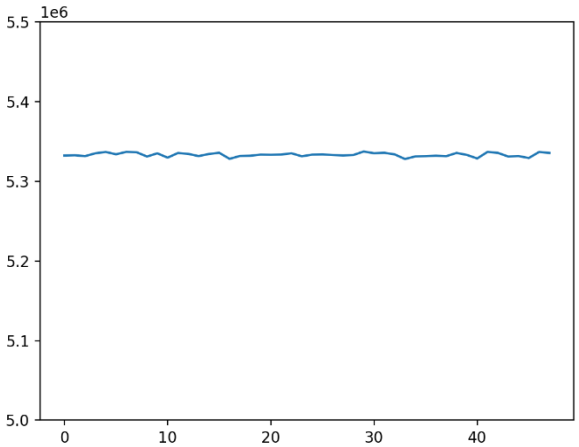


图 3 Extended Olken 结果元组与其抽样次数的关系

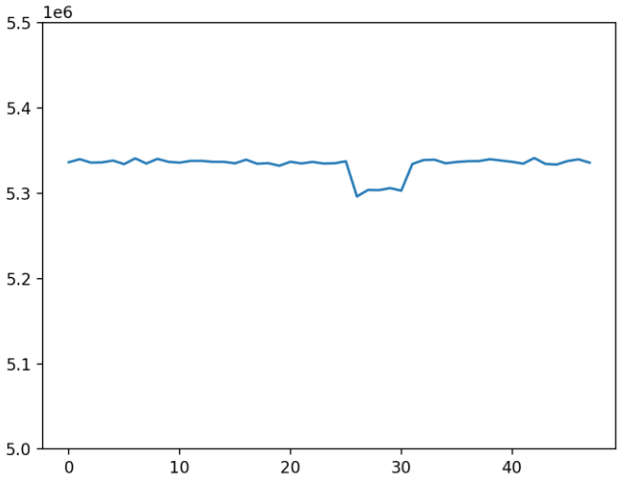
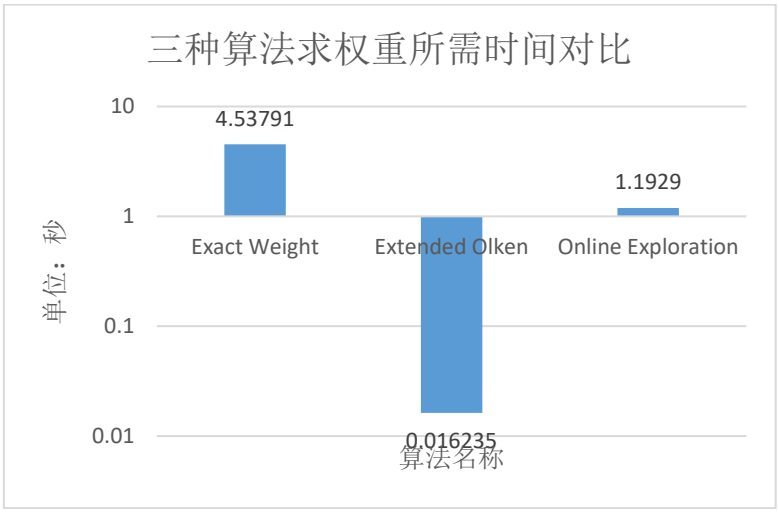
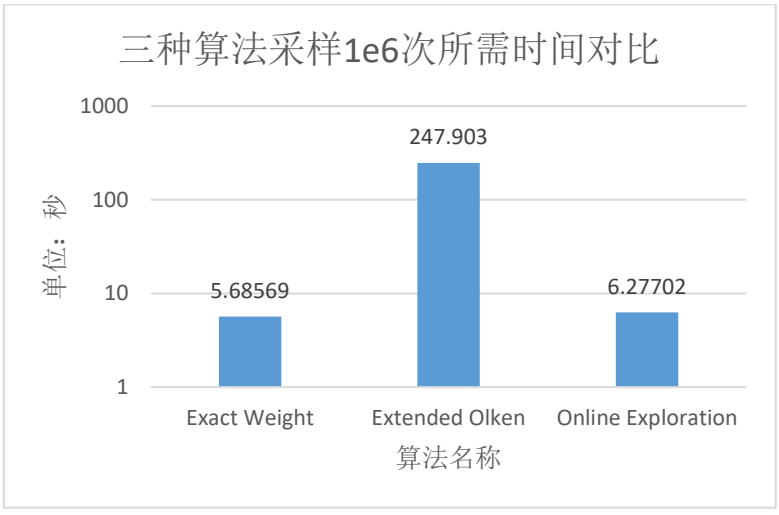
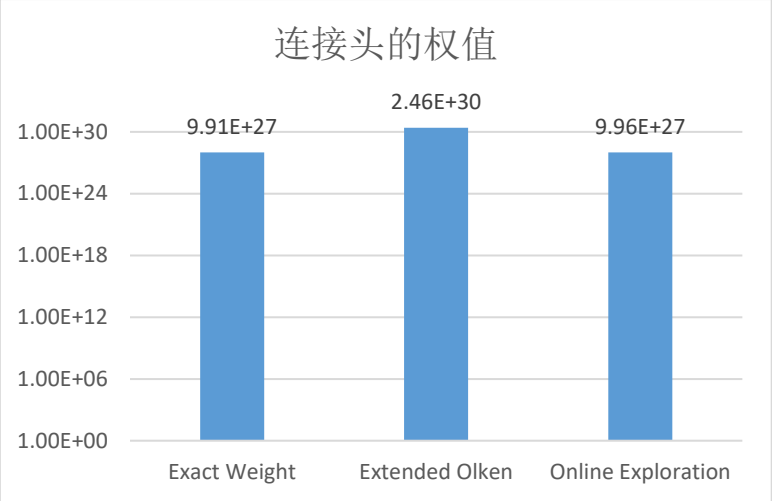
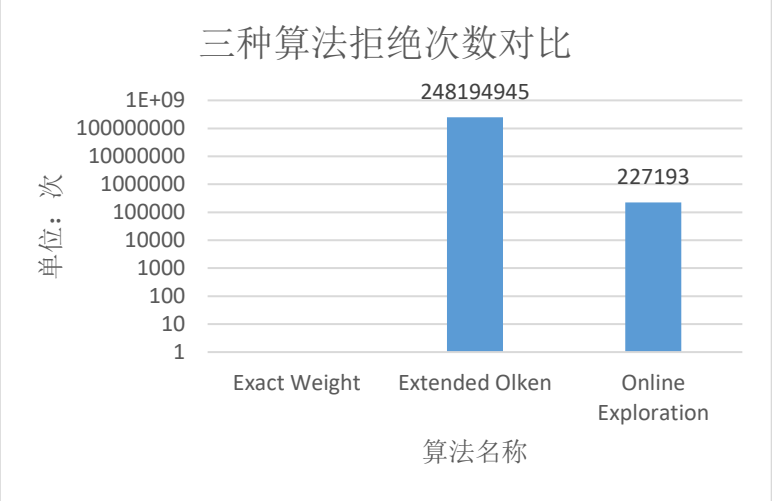


图 4 Online Exploration 结果元组与其抽样次数的关系









五、结果观察与分析

1、 首先我们会注意到的是，预处理权重方面，EO 是远远胜于 EW 和 OE 方法的，而 OE 方法也大概只用了 EW 方法的四分之一，效率还是非常不错的。
2、 采样效率方面，OE 方法比 EW 方法略慢，而 EO 方法更是慢了几十倍。通过对比采样被拒绝次数，我们也发现符合类似的规律，说明采样的效率直接受拒绝率的影响。
3、 采样效果方面，我们会发现 EW，EO，RS 每个元组采样次数相差无几，均为均匀采样。而 OE 方法有一小部分略微比其他元组的采样次数要少，这是因为我在编写这部分代码是没有计算置信区间，也就没有更新其上上限，从而可能会出现 $\exists 0 \leq i \leq n-1, \exists t_i \in R_i, W(t_i) \leq W(t_i \rtimes R_{i+1})$ ，这就不符合了通用算法框架的条件，从而导致采样效果有轻微偏离均匀采样的情况，但是实际上偏差率为 1% 左右。
4、 将连接头的权重输出我们可以发现 EO 的连接头的权重比另外两个算法权重高三个数量级。而 OE 与 EW 在同一个数量级，且比 EW 略大，说明权重的估计值越大，其拒绝率越高。

六、实验过程中最值得说起的几个方面

我所生成的数据表文件缺陷还有很多，并不能很好的展示这三种算法性能的优劣，而且在代码编写上纠结了许多编写上以及实验上的方式，但是实际上可以与同学交流来解决我的

问题，节省精力和时间。