程序设计实习（实验班-2024春）
Sparse Recovery的数据流算法

授课教师：姜少峰

助教：冯施源 吴天意

Email: shaofeng.jiang@pku.edu.cn

从近似计算直径的数据流算法说起

- 我们学过如何在线性时间对直径做$(1+\epsilon)$-近似
- 是否可以做成大数据算法，例如数据流算法?
- 我们将介绍基于sparse recovery的方法，来实现：
    - $O(\epsilon^{-d}\text{poly}\log n)$空间的数据流算法来给出直径的$(1+\epsilon)$-近似
- 我们还将介绍与sparse recovery有关的其他几个算法

线性时间$(1+\epsilon)$-近似直径

T可通过选取任意点u，求u到最远点的距离得到（见第一讲）

即满足$1/2 \cdot \mathrm{diam}(P) \le T \le \mathrm{diam}(P)$

- 先O(n)时间找一个直径的2-近似值T

可以是任意确定点

- 作$\ell := \epsilon \cdot T/\sqrt{2}$的网格，并round到中心
  - 因此每个点移动了$\le \sqrt{2}\ell \le \epsilon \cdot \mathrm{diam}(P)$
- 因此新点集P'满足

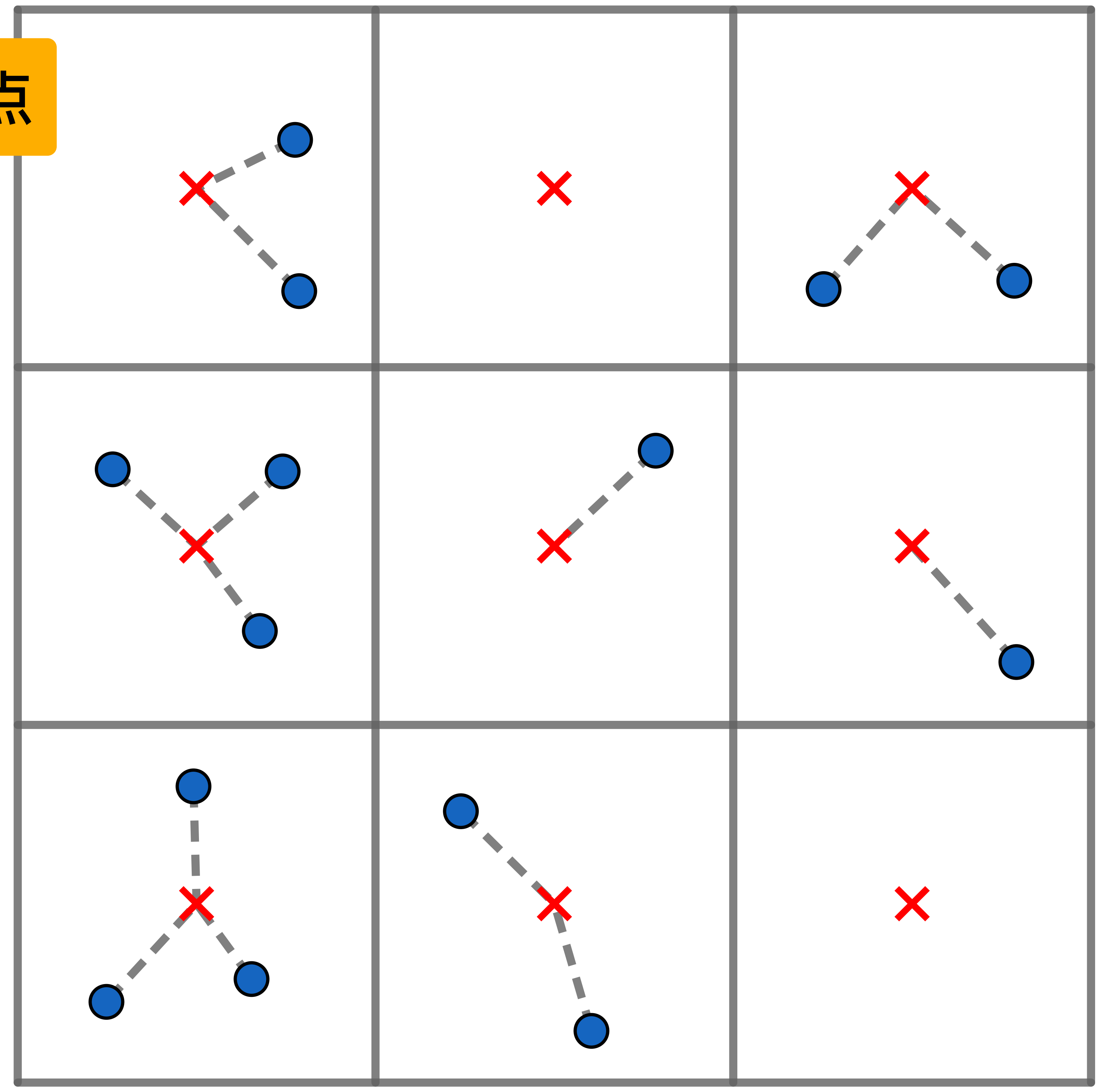
可在$O(nd \log n)$时间构造P'

$$\mathrm{diam}(P') \in (1 \pm \epsilon) \cdot \mathrm{diam}(P)$$

- 算法：在$|P'|$上暴力求直径，复杂度$|P'|^2 \cdot d$

P'所有点都在$\mathrm{diam}(P') \times \mathrm{diam}(P')$大方格内，小方格$\ell \ge \Omega(\epsilon \cdot \mathrm{diam}(P))$，故$|P'| \le (O(1/\epsilon))^2$

总复杂度：$O(nd \log n) + O(1/\epsilon)^d$



数据流算法模型

$[n]=\{1,\ldots,n\}$，不失一般性因为有限domain总可以映射到$[n]$

- 在某个domain上的数据，不失一般性设为[n]，以数据流的方式给出
  - 数据流是一个插入/删除domain元素的操作序列，只支持单次、顺序访问
  - 例如：ins 1，ins 2，ins 2，ins 3，del 1后，得到的数据集是$\{2,2,3\}$

一般允许有重点

- 在数据流结尾，在当时的数据集上进行某些查询操作

一般不需要实时进行，只需要在结尾进行一次查询

要求：使用亚线性、尽量少的空间，时间是第二考虑但追求均摊$\text{poly}\log n$

举例：2维欧氏点集直径问题的数据流设定

这就是数据的domain，例如int坐标点就是int乘以int的区域
可以不失一般性映射到$[n]$（$n = \Delta^2$）

- 设所有可能的数据点都在一个$\Delta \times \Delta$的区域
- 数据流：一系列点坐标的插/删，例如

一般地，允许有重点

  - ins (1, 1), ins(1, 0), ins(2, 0), del(1, 0), ins (3, 4)后得到$\{(1,1), (2,0), (3,4)\}$
- 当数据流结束后，给出点集直径（的估计）
  - 上面的例子就是$\{(1,1), (2,0), (3,4)\}$这个点集的直径

数据集的等价表达：频数向量

- 数据集可以用每个元素出现的频数的频数向量$\mathbf{x}$进行等价表达
  - $\mathbf{x}$每维对应一个数据domain上的点，值等于该点出现了多少次
  - 例如对于int型的domain，$\mathbf{x}$就是2^{32}长度
  - 刚刚的例子的$\{2,3\}$对应的频数向量中，只在2、3对应维度上是1，其他是0

一般假设：频数向量每一维的最大绝对值，也就是$\|\mathbf{x}\|_\infty$，是$\mathrm{poly}(n)$的

数据的Support

我们一般用p代表一个数据点

即在数据流结束时仍存在的元素

- 对于频数向量$\mathbf{x}$，令$\text{supp}(\mathbf{x}) := \{p : x_p \neq 0\}$代表$\mathbf{x}$中非0的坐标
- 考虑$\text{supp}(\mathbf{x})$的一大作用是忽略重复点，并且很多操作都是定义在supp上
- 相关概念：ℓ_0范数，定义为$\|\mathbf{x}\|_0 := |\text{supp}(\mathbf{x})|$
    - 频数向量的ℓ_0范数代表数据有多少不同的元素

重要工具：Sparse Recovery

可以解决直径问题

技术上说，这里允许负频数，并且只要不是0频数，都算作support里面

- 复习：称一个频数向量$\mathbf{x}$是k-sparse的若$\|\mathbf{x}\|_0 \leq k$
- Sparse recovery是一个数据流算法，给定一个参数k：
  - 检测数据流（的频数向量）是否是k-sparse的

    回答Yes/No
  - 如果Yes，就把$\mathrm{supp}(\mathbf{x})$完全恢复出来

    即输出一个集合，等于$\mathrm{supp}(\mathbf{x})$，一共至多k个元素

结论：存在一个高概率成功的$O(k \cdot \mathrm{poly} \log n)$空间的sparse recovery算法

更新时间$\mathrm{poly} \log n$
查询时间$O(k \cdot \mathrm{poly} \log n)$

n是domain大小，也就是频数向量的长度/维数

利用Sparse Recovery解决数据流直径问题

大体思路

只插入也是容易的，但是带删除呢？

2-近似：从任意点出发，找最远

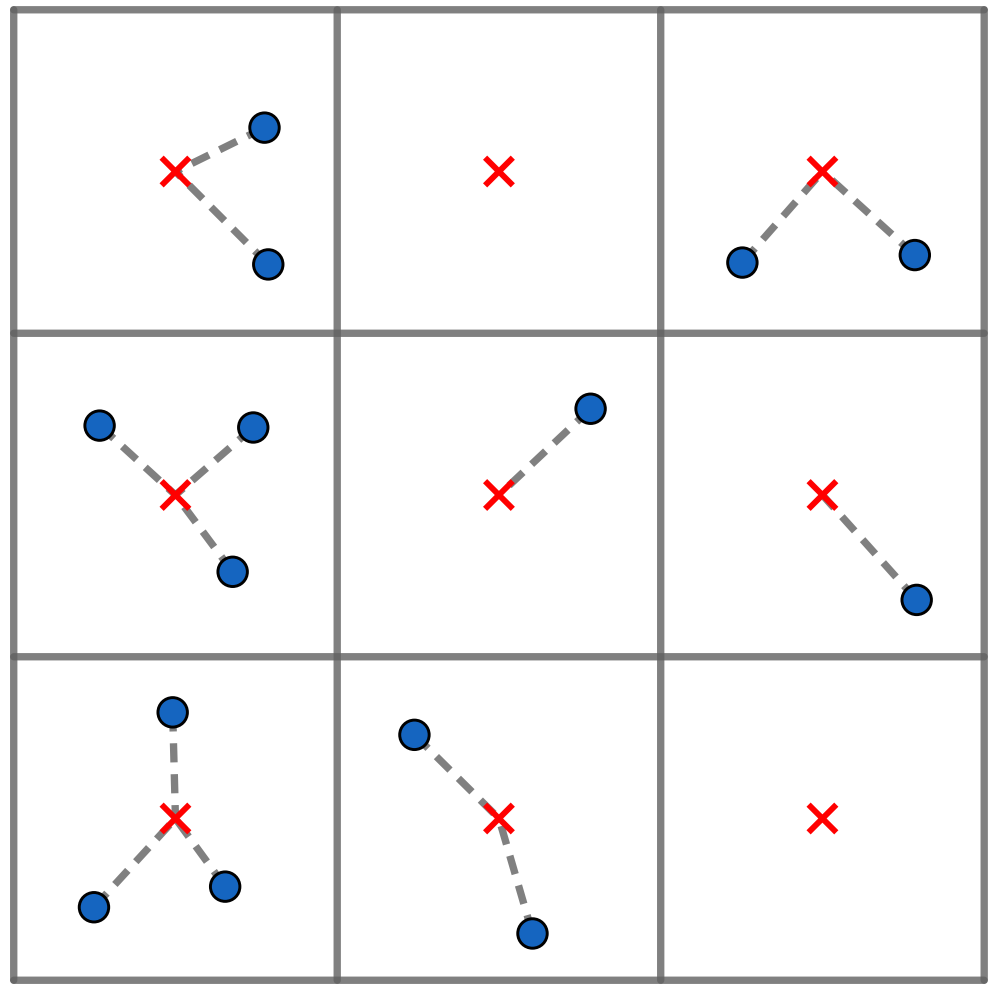
- 在离线算法中，需要先对直径有一个$O(1)$-近似，但很难在带删除的数据流做到
- 新的idea：“穷举”/“猜”/二分查找直径的值，然后验证猜测是否正确
- 性质：若猜的$D \geq \mathrm{diam}(P)$，数据点移到ϵD格点后“非空”格点有$(O(1/\epsilon))^d$个

设点集是P，$\mathrm{diam}(P)$是直径

可以采用$k = \epsilon^{-O(d)}$的
sparse recovery暴力恢复

- 原因：
  - 每个点移动量至多$O(\epsilon D)$，移动后点击直径至多$O(D)$
  - 一个$O(D) \times O(D)$的方可以容纳多少$\epsilon D \times \epsilon D$的小方？

$\epsilon^{-O(d)}$



算法：利用Sparse Recovery解决数据流直径问题

以2倍为步长穷举/猜测直径的值

- for $i = 0, \ldots, O(\log(\Delta))$
  - 维护domain在$\Delta \times \Delta$上的k-sparse recovery structure $\mathcal{S}_i$，参数$k = O(1/\epsilon)^d$
  - 当数据流插/删点p时，找到p所在的$\epsilon \cdot 2^i$格点中心p'，将p'从$\mathcal{S}_i$插入/删除
- 数据流结束时，询问每一个$\mathcal{S}_i$是否k-sparse
  - 找到i值最小的返回Yes的$\mathcal{S}_i$，返回恢复的点集并求该点集直径作为结果返回

刚刚看到，猜测 $\geq \mathrm{diam}(P)$时都会返回Yes
因此需要找符合条件的最小i值来找到 $\approx \mathrm{diam}(P)$的猜测
事实上，可能会找到更小的i，但这只会让解更加精确

时空复杂度分析?

更新时间$\text{poly}\log n$，
查询$O(k\cdot\text{poly}\log n)$

结论：存在一个高概率成功的$O(k\cdot\text{poly}\log n)$空间的sparse recovery算法

- 使用了$O(\log(\Delta))$个$k=O(1/\epsilon)^d$的sparse recovery结构，domain大小是Δ^2
- 总共空间是$O(1/\epsilon)^d\cdot\text{poly}\log(\Delta)$
- 更新时间是$\text{poly}\log\Delta$，查询时间$O(1/\epsilon)^d\cdot\text{poly}\log(\Delta)$

数据流Sparse Recovery

思路

- 先考虑$k = 1$
  - 测试是否有至多1个的不同元素
  - 若上述判断为Yes，则只有一个不同元素，设计算法将它恢复出来
- 对于一般的k
  - 将元素用足够大、足够多的随机哈希映射到$O(k)$个bucket里

使得k个元素的每一个都能找到一个无冲突/只含有它自己的bucket，从而实现1-sparse recovery

$k=1$：先假设必有至多一个不同元素，如何恢复?

- 假设恰有一个元素，一个确定性算法：

例如插/删p时，$c_1 := c_1 \pm 1$，$c_2 := c_2 \pm p$

设$\mathbf{x} \in \mathbb{Z}^n$是频数向量，维护$c_1 := \sum_{p=1}^{n} x_p$，$c_2 := \sum_{p=1}^{n} p \cdot x_p$

- 最后计算$p = c_2/c_1$就是唯一出现的元素编号，c_1就是出现次数

一个有趣的事实：该算法甚至适用于负频数

- 遗留情况：$c_1 = 0$，此时在1-sparse的前提下，必然说明$\mathbf{x} = 0^n$

如何检测数据流有多于1个不同元素?

一个错误算法：仍维护$c_1 := \sum_{p=1}^{n} x_p$，$c_2 := \sum_{p=1}^{n} p \cdot x_p$

寄希望于多于1个元素时$p' = c_2/c_1$除不尽，但这个未必成立

利用fingerprinting：设随机$r \in \{0,\ldots,q-1\}$其中$q = \mathrm{poly}(n)$是素数

多维护一个$c_3 := \sum_{p=1}^{n} x_p \cdot r^p \mod q$

远大于n和每个x_p的最大值即可

一般假设：频数向量每一维的最大绝对值，也就是$\|\mathbf{x}\|_\infty$，是$\mathrm{poly}(n)$的

检测算法

设随机$r \in \{0,\ldots,q-1\}$其中$q=\mathrm{poly}(n)$是素数

算法：维护$c_1 := \sum_{p=1}^{n} x_p$，$c_2 := \sum_{p=1}^{n} p \cdot x_p$，以及$c_3 := \sum_{p=1}^{n} x_p \cdot r^p \mod q$

- 如果c_2不能被c_1整除，则返回No
- 否则，令$p' = c_2/c_1$，若$c_3 = c_1 \cdot r^{p'} \mod q$返回Yes，否则No

当频数向量确实1-sparse时，该算法必然返回Yes（与r的随机选取无关）

但如果不是1-sprase呢?

概率分析

对于对素数q取模也成立

设$g(t) := \sum_{p=1}^{n} x_p \cdot t^p - c_1 \cdot t^{p'} \mod q$，则$g(t)$是$\leq n$次多项式，有至多$n$个零点

$g(r) = 0$对应的就是$c_3 = c_1 \cdot r^{p'}$返回Yes的情况

如果数据流不是1-sparse的，$\Pr[g(r) = 0] \leq n/q \leq 1/\mathrm{poly}(n)$

因此：新算法可以大概率检测是否$\|\mathbf{x}\|_0 > 1$

推广到k-sparse：假定数据集是k-sparse的，如何恢复?

- 不妨设k个不同元素是$[k]$
- 先考虑一个随机哈希$h : [n] \to [2k]$
  - 考虑$p \in [k]$：$\Pr[\forall q \neq p \in [k], h(p) \neq h(q)] = (1 - 1/(2k))^{k-1} \geq 0.5$

p所在bucket只含有p，即无冲突的概率

- 因此存在与p的哈希冲突的概率$\Pr[\exists q \neq p \in [k], h(p) = h(q)] \leq 0.5$

利用多个hash

“多次试验”

然后考虑$T = O(\log k)$个独立的hash $\{h^{(i)} : [n] \to [2k]\}_{i=1}^{T}$

我们想要：Pr[对所有$p \in [k]$都存在一个无冲突的$h^{(i)}$] > 0.5

- 对某确定的$p \in [k]$，所有T个hash $h^{(i)}$上p都有冲突的概率 $\leq 0.5^T \leq 1/(2k)$
- 推出：存在$p \in [k]$使得所有T个hash $h^{(i)}$上p都有冲突概率 ≤ 0.5

union bound

反面事件就是我们想要的：对所有$p \in [k]$都存在一个无冲突的$h^{(i)}$的概率 > 0.5

完整算法：假定数据是k-sparse的

初始化：

- 设置$T = O(\log k)$个独立的随机hash $\{h^{(j)} : [n] \to [2k]\}_j$
- 对任何$i \in [T]$，$j \in [2k]$，维护一个1-sparse recovery $\mathcal{S}_{ij}$

数据流插/删元素$p \in [n]$时：对所有$i \in [T]$，令$j = h^{(i)}(p)$，在$\mathcal{S}_{ij}$插/删p

数据流结束时：找到所有的1-sparse的$\mathcal{S}_{ij}$，返回恢复结果的并集

注意：我们可以将1-sparse的元素ID和元素出现频数都恢复出来！

成功概率就是上页给出的0.5；
如何提高成0.99?

遗留问题：如何检测有多于k个不同元素?

先当作频数向量是k-sparse的，运行上页的算法，最后仍会恢复出一些元素来

如果所有1-sparse和随机哈希都“成功”了
则这些恢复出来的元素必定是“真”元素，且如果真是k-sparse则一定都能恢复出来

如果恢复出超过k个不同元素，那就不是k-sparse的

检测一个数据流最后是不是空的，
可以使用1-sparse recovery

否则，把这些元素从数据流删除，检测删除后的数据流是不是空的

如果原来是k-sparse的，那么就恢复成功了，删除这些元素后会是一个空数据流

如果原来不是k-sparse的，那么删除完了就不会是空数据流

完整算法：检测是否是k-sparse

- 设之前提到的假定k-sparse并进行恢复的算法叫做$\mathscr{A}$
- 在输入数据流上同时运行/维护算法$\mathscr{A}$和一个1-sparse recovery算法$\mathscr{B}$
- 数据流结束时查询$\mathscr{A}$，并设频数向量$\hat{\mathbf{x}}$是$\mathscr{A}$的输出
- 将“删除$\hat{\mathbf{x}}$”这一操作继续传给$\mathscr{B}$
- 此时查询$\mathscr{B}$，看是否数据流是空的，如果是空的则返回Yes否则No

需要一个独立的$\mathscr{B}$来做1-sparse检测，因为$\hat{\mathbf{x}}$与$\mathscr{A}$的随机性相关，只用继续从$\mathscr{A}$删除$\hat{\mathbf{x}}$会导致$\mathscr{A}$的概率保证全部失效

今天介绍的算法其实是Linear Sketch

- 称一个（数据流）算法是linear sketch，如果可以把算法看作如下形式：
  - 设$\mathbf{x}$是频数向量，则算法在数据流下维护的是一个线性操作后的结果$\mathbf{Ax}$
  - 算法在回答查询时，仅利用$\mathbf{Ax}$，即可以写成$f(\mathbf{Ax})$

$\mathbf{A}$可以是随机的

- 我们介绍的sparse recovery算法是linear sketch
  - 1-sparse recovery维护的是若干求和
  - k-sparse recovery一旦哈希确定了，就也是若干求和

这些哈希可以理解成A，可以是随机的

Linear Sketch的优点

- 假设对数据流P_1和P_2分别维护了linear sketch $\mathcal{S}_1, \mathcal{S}_2$
- 则$\mathcal{S}_1 + \mathcal{S}_2$就是$P_1 \cup P_2$的sketch

注意：想要可合并，还需要两个sketch的随机性是一致的，例如哈希要用同一个

- 同理也可以做减法：
    - 例如P_1是前n个元素，P_2是前10个元素，那么$\mathcal{S}_1 - \mathcal{S}_2$就是[11, n]的元素
- 这种可合并性对于大数据十分友好：分布式对每个数据片段执行之后合并汇总

因此我们介绍的算法虽然看上去是单机算法，但可以很容易对数据分组运行后同时运行在多机，得到整个大数据的sketch

与压缩感知的一些对比

- 都是linear sketch/linear measurements
- 我们可以理解成对于频数向量的压缩感知，但频数向量是通过数据流给出的
  - 本质上，我们也是以$\mathbf{b} = \mathbf{A}\mathbf{z}$的方式来进行恢复的

对我们来说z就是频数向量

- 今天介绍的是更适合大数据设定的一种方法
  - 直接的压缩感知方法无法有效在数据流实现

作业十九：欧氏点集直径的数据流算法

- http://cssyb.openjudge.cn/24hw19/
- 分值：4分
- 截止日期6月21日

扩展：最小包围球的数据流算法

$(1+\epsilon)$-近似，ϵ^{-d}空间

- 最小包围球 (MEB)：给定数据集$P \subset \mathbb{R}^d$，找到中心c以及最小的$r \geq 0$，使得

$B(c,r) := \{y : \|c-y\| \leq r\}$包含所有$P$中的数据点

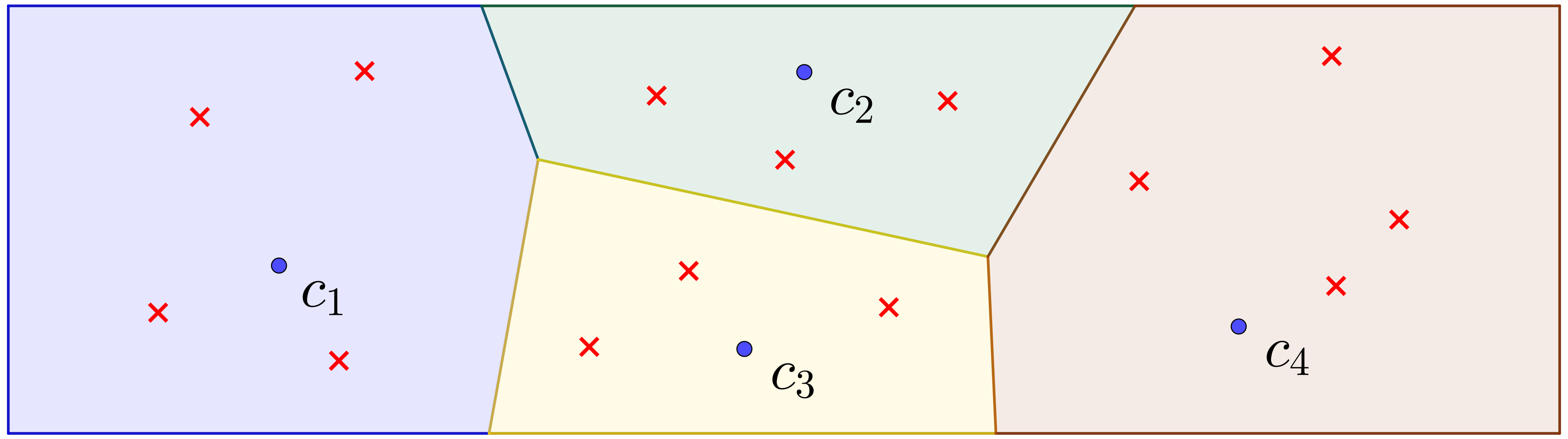
- 可以采用类似的算法，猜r，然后数据点移动到ϵr-格点上，在格点上求MEB
  - 正确性：对于"正确"的猜测r，数据点移动量 $\leq \epsilon \mathrm{OPT}$
  对结果也是这么大的影响
  - 空间：对于"正确"的猜测r，数据点包含在半径r的球里，含有至多ϵ^{-d}个格点
- 需要讨论r"猜大"/"猜小"怎么操作的问题

扩展：k-Center Clustering的数据流算法

- k-center clustering：找k个数据点，记为$C \subseteq P$，使到最远数据点的距离最小

$\mathrm{dist}(p, C) = \min_{c \in C} \|p - c\|$

$$\min_{C} \max_{p \in P} \mathrm{dist}(p, C)$$



- 与MEB的关系：等价于找最小的半径r，使得k个半径r的球能覆盖数据集P

球心构成的集合就是C

算法：$(1+\epsilon)$-近似，$k\cdot\epsilon^{-d}$空间

- 算法与MEB非常类似，也是猜r，移动到ϵr格点，然后在格点上求k-center
  - 注意：求格点上的k-center的$(1+\epsilon)$近似需要指数时间（假设$\mathrm{P}\neq\mathrm{NP}$）
  - 但空间依然可控，所以还是“好”的数据流算法
- 正确性：对于猜测$r\approx\mathrm{OPT}$，每个数据点移动量$\epsilon\mathrm{OPT}$

对结果也是这么大的影响

- 空间：数据点可以被k个半径是r的球覆盖，每个里面会产生ϵ^{-d}格点
  - 总共就是$k\cdot\epsilon^{-d}$空间

数据流ℓ_0-采样

相关问题：ℓ_0-采样

- 刚刚介绍的sparse recover是当support比较小的时候全面恢复support的方法
- 当support比较大时，我们希望通过采样来了解support的情况/计算统计量

ℓ_0-采样是一个数据流算法，返回一个$\mathrm{supp}(\mathbf{x})$上的均匀采样$p$

$$\forall q \in \mathrm{supp}(\mathbf{x}), \quad \Pr[p = q] = \frac{1}{|\mathrm{supp}(x)|}$$

ℓ_0-采样

并且是linear sketch，支持合并/求差！

- 结论：存在一个使用空间$\mathrm{poly}\log n$的ℓ_0-采样算法，以$1-1/\mathrm{poly}(n)$概率成功
- 一些重要note：

  可以理解成反复询问也只会返回同一个样本

  - 算法确实是输出了一个均匀采样，但是反复运行算法无法生成新的独立采样
  - 要想生成独立采样，需要运行另一个（独立的）ℓ_0-采样算法

ℓ_0采样可以用来实现Sparse Recovery

- 可以用来实现sparse recovery

又叫coupon collection

- 事实：有m种盲盒均匀分布，开$T = O(m \log m)$个可以大概率凑齐所有m种
  - 第i种T次全都不出现概率 $= (1 - 1/m)^T \leq 1/\text{poly}(m)$
- 因此：对于k-sparse数据流，只需要独立运行$O(k \log k)$个ℓ_0-采样就可以恢复
  - 使用的空间与sparse recovery算法基本一致（差一些$\text{poly} \log n$项）

ℓ_0-采样的实现思路

- 一个对于supp($\mathbf{x}$)的均匀采样可以这样得到：
  - 设 $|\mathrm{supp}(\mathbf{x})| = k$

    即抛一枚正面朝上概率1/k的硬币
  - 将supp($\mathbf{x}$)上所有元素以$1/k$概率subsample
  - 在“活下来”的元素中进行均匀采样
- “活下来”的元素有常数概率只有1个，可以调用1-sparse recovery！

ℓ_0-采样的实现

由于我们不预先知道 $|\mathrm{supp}(\mathbf{x})|$，需要穷举猜测

$\mathcal{S}_i$维护的是以2^{-i}概率存活的点

- 初始化：对$i = 0,\ldots,O(\log n)$，维护一个1-sparse recovery $\mathcal{S}_i$
  - 并设有随机哈希$h^{(i)} : [n] \to \{0,1\}$使得$\forall p \in [n]$，$\Pr[h(p) = 1] = 2^{-i}$

可以使用universal hash来省空间

- 插/删元素$p \in [n]$时：
  - for $i = 0,\ldots,O(\log n)$，若$h^{(i)}(p) = 1$则将p插入$\mathcal{S}_i$
- 当数据流结束时：找到（任意）一个返回Yes的$\mathcal{S}_i$，返回恢复出来的元素

Yes = 确实是1-sparse

可以将整个过程独立重复若干次来提升成功概率

这也同时可以返回频数

ℓ_0-采样的应用

图数据流上求连通分量

- 图数据流设定：输入是一个无权无向图边集的数据流
  - 例如ins(1, 2), ins(2, 3), ins(1, 3), del(1, 2)

两个数字代表一条边的两个端点

- 设n为图上顶点个数，m为边数

因此$m < n^2$

[Ahn-Guha-McGregor, SODA 12]

每次更新的时间是$\text{poly}\log(n)$

结论：存在一个使用$\tilde{O}(n)$空间**精确计算**图的**连通分量**的数据流算法

离线做法BFS、DFS需要$O(m+n)$空间，带删除就更加困难

$\tilde{O}(f) := O(f \cdot \text{poly}\log f)$

输出每个点从属于的连通分量的编号

一个离线算法

输入： $G=(V,E)$

初始化： 每个顶点$v_i \in V$作为一个单独连通分量$S_i := \{v_i\}$

while 上轮连通分量情况发生变化

称$e=(u,v)$是跨越S_i的边，若$u \in S, v \notin S$

对每个分量S_i找任何一个跨越边e_i，将这些跨越边加入后更新连通分量$\{S_i\}_i$

算法的while循环至多运行$O(\log n)$轮，因为每轮每个S_i都会被合并，大小至少会double

这个算法可以给出一个生成森林

数据流实现：思路

while 上轮连通分量情况发生变化

  对每个分量S_i找任何一个跨越边e_i，将这些跨越边加入后更新连通分量$\{S_i\}_i$

- 由于我们可以承受$\tilde{O}(n)$空间，每轮while的连通分量$\{S_i\}_i$是可以存下的
- 那么核心问题变成了：对每个S_i，如何用较少空间找到跨越边e_i?
- 大体思路是设计一种很特殊的ℓ_0采样的“频数向量”，使support = 跨越边集

巧妙利用频数向量来得到“跨越边集”

$\mathbf{x}^v$的每一维对应一条可能的无向边

对每个顶点$v \in V$，定义频数向量$\mathbf{x}^v \in \mathbb{Z}^{\binom{V}{2}}$

$$\mathbf{x}^v(u,v) := \begin{cases} 1 & (u,v) \in E, u \le v \\ -1 & (u,v) \in E, u > v \\ 0 & \text{otherwise} \end{cases}$$

假设顶点按1 ~ n编号
注意，这里面只有与v邻接的边对应的位置才可能非0

即，同一条边(u,v)，设$u \ge v$，在$\mathbf{x}^u$上是+1，在$\mathbf{x}^v$上是-1

重要性质：设$S \subseteq V$，则$\mathbf{x}^S := \sum_{v \in S} \mathbf{x}^v$的$\mathrm{supp}(\mathbf{x}^S)$只含有跨越$S$的边

不妨先考虑$S = \{p,q\}$只有两个点，只有s,t是内部边
则$\mathbf{x}^S(p,q) = \mathbf{x}^p(p,q) + \mathbf{x}^q(p,q) = 0$，同一条边正负抵消

数据流算法

$$\mathbf{x}^v(u,v) := \begin{cases} 1 & (u,v) \in E, u \leq v \\ -1 & (u,v) \in E, u > v \\ 0 & \text{otherwise} \end{cases}$$

初始化：每个点v维护一个频数向量是$\mathbf{x}^v$的ℓ_0-采样结构$\mathscr{L}_v$

维护：每插/删一条边$e = (u,v)$，更新$\mathscr{L}_u$和$\mathscr{L}_v$

设$u \leq v$，则$\mathscr{L}_u$对应的$\mathbf{x}^u(u,v) := \mathbf{x}^u(u,v) - 1$
$\mathbf{x}^v(u,v) := \mathbf{x}^u(u,v) + 1$

查询：当需要得到算法运行中某个S_i的跨越边时，计算$\mathscr{L}_S := \sum_{v \in S_i} \mathscr{L}_v$

然后从$\mathscr{L}_S$上生成一个采样，这个采样就是S_i的某个跨越边e_i

利用ℓ_0-采样结构是linear sketch
可合并的特点

更进一步：图数据流最小生成树

- 输入是一个带权无向图边集的数据流
  - 例如ins(1, 2, 5), ins(2, 3, 4), ins(1, 3, 6), del(1, 2, 5)

前两维是两个端点，第三维是边权

- 设n为图上顶点个数，m为边数

设边权范围在$\mathrm{poly}(n)$内

每次更新的时间是$\mathrm{poly}\log(n)$

结论：存在一个使用$\tilde{O}(n/\epsilon)$空间$(1+\epsilon)$-近似MST的数据流算法

$\tilde{O}(f) := O(f \cdot \mathrm{poly}\log f)$

思路

- 首先，我们需要介绍一个特别适合于大数据的MST离线算法：Borůvka算法
  - Borůvka算法是1926年提出的，基本上是已知最早的求MST的算法
- 我们的算法就是Borůvka算法的数据流实现
  - 该算法与求连通分量的方法有相似之处

Borůvka算法

输入：$G=(V,E)$，边权函数$w:E\to\mathbb{N}$

初始化：每个顶点$v_i\in V$作为一个单独连通分量$S_i:=\{v_i\}$

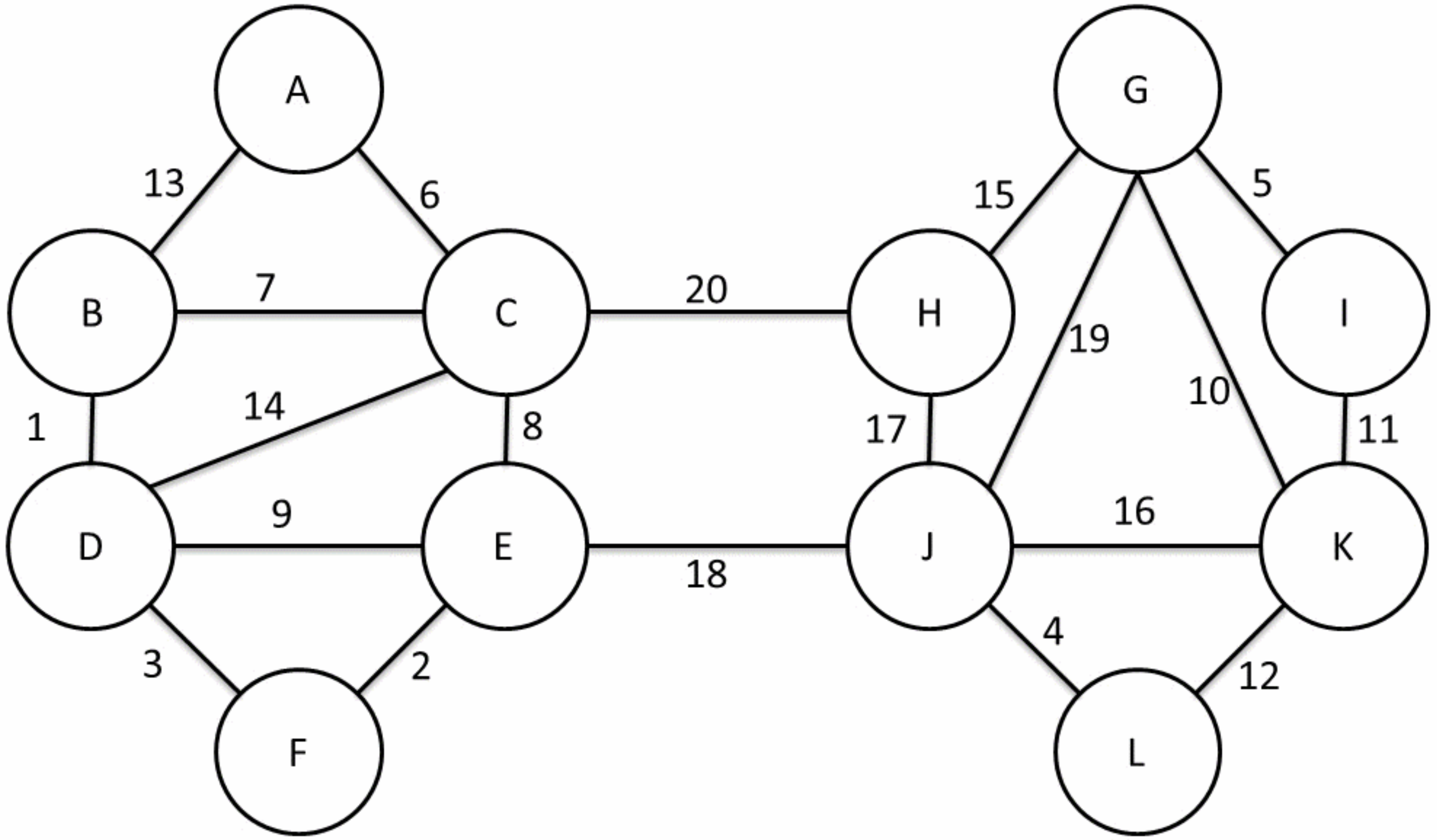
while 当前仍有超过一个分量

称$e=(u,v)$是跨越S_i的边，若$u\in S, v\notin S$

对每个S_i找跨越S_i的最小权边e_i，将$\{e_i\}_i$由小到大加入、成环则丢弃，更新$\{S_i\}_i$

事实上可证明并不会成环

视频演示



数据流实现思路

- 类似于求连通分量，重点也是找跨越某个集合S的跨越边，但此时要求最小权
- 大体思路：
  - 将边数据流划分成子数据流，每个对应一种边权
  - 每个子数据流维护一套求连通分量用的sketch
  - 查询时：从小到大穷举边权，找到最小的有跨越S的边的边权，并返回一条边
- 问题：边权种类可能有$O(m) = O(n^2)$种，每种维护一套sketch需要$\tilde{\Omega}(m)$空间

解决方案：离散化边权

- 离散化边权：将边权向上round到最近的$(1+\epsilon)$的方幂
  - 这步可以在数据输入的时候顺带直接完成，因此可以不失一般性作出该假设
- 这样一共有$\log_{1+\epsilon}(\text{poly}(n)) \leq O(\log(n)/\epsilon)$种边权

完整算法

同连通分量算法中的$\mathscr{L}_v$，只不过对每个i有一个独立的copy

- 初始化：设$L := O(\log(n)/\epsilon)$，对$i = 0,\ldots,L$和$v \in V$初始化一个$\mathscr{L}_v^{(i)}$
- 插/删(u, v, w)：设w'是round后的边权，$j := \log_{1+\epsilon} w'$，更新$\mathscr{L}_u^{(j)}$、$\mathscr{L}_v^{(j)}$
- 查询：模拟Borůvka算法，其中当要寻找跨越S的边时
  - for $j = 0,\ldots,L$
    - 计算$\mathscr{L}_S^{(j)} := \sum_{v\in S} \mathscr{L}_v^{(j)}$，用$\mathscr{L}_S^{(j)}$采样一条边，若存在则返回该边，否则继续

课程评估