Algorithmic Techniques for Independent Query Sampling

Yufei Tao

Chinese University of Hong Kong

S := the input set

Query mechanism in database systems:

- Given a predicate q
- Report S_q := the set of elements in S satisfying q.

S := the input set

Query mechanism in database systems:

- Given a predicate q
- Report S_q := the set of elements in S satisfying q.

Works well until S becomes "big".

- When $|S| = 10^6$, 1% selectivity means 10000.
- When $|S| = 10^9$, 1% means 10^7.

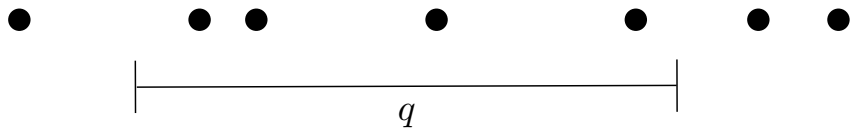
Remedy: Sampling

An old technique since the 90's of the last century.

- Given a predicate q and a **sample size** $s \geq 1$
- Return a **size-s sample set** of S_q.

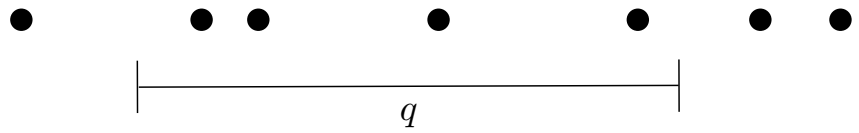
Example: Range Sampling

$S :=$ a set of real values
$q := [x, y]$
$S_q := S \cap q$
$s := 10$



Example: Range Sampling

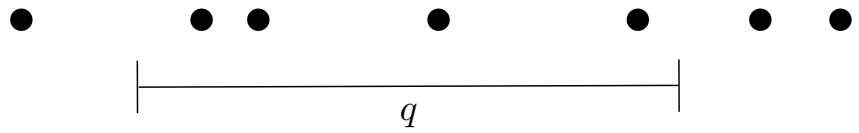
$S :=$ a set of real values
$q := [x, y]$
$S_q := S \cap q$
$s := 10$



- **WR (with replacement)**
  repeat 10 times: take an element from S_q uniformly at random.

Example: Range Sampling

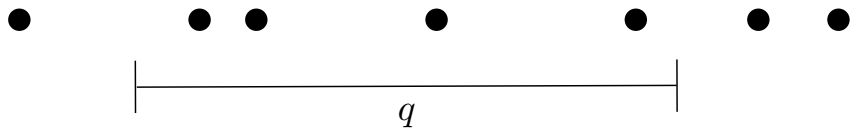
$S :=$ a set of real values
$q := [x, y]$
$S_q := S \cap q$
$s := 10$



- **WR (with replacement)**
  repeat 10 times: take an element from S_q uniformly at random.
- **WoR (without replacement)**
  any of the $\binom{|S_q|}{10}$ subsets of S_q is returned with the same probability.

Example: Range Sampling

$S :=$ a set of real values
$q := [x, y]$
$S_q := S \cap q$
$s := 10$



- **WR (with replacement)**
  repeat 10 times: take an element from S_q uniformly at random.
- **WoR (without replacement)**
  any of the $\binom{|S_q|}{10}$ subsets of S_q is returned with the same probability.
- **Weighted**
  - Each element $e \in S_q$ has a **weight** $w(e) > 0$
  - repeat 10 times: take an element from S_q with a probability proportional to its weight.

Issue: Sample results of different queries can be correlated!

Issue: Sample results of different queries can be correlated!

$S :=$ a set of real values

Issue: Sample results of different queries can be correlated!

$S :=$ a set of real values

First query:

$q := [100, 200]$ and $s := 1$

Issue: Sample results of different queries can be correlated!

$S :=$ a set of real values

First query:

$q := [100, 200]$ and $s := 1$

Return a uniformly random element $e \in S_q := S \cap q$

Issue: Sample results of different queries can be correlated!

$S :=$ a set of real values

First query:

$q := [100, 200]$ and $s := 1$

Return a uniformly random element $e \in S_q := S \cap q$

Second query:

$q := [100, 200]$ and $s := 1$

Issue: Sample results of different queries can be correlated!

$S :=$ a set of real values

First query:

$q := [100, 200]$ and $s := 1$

Return a uniformly random element $e \in S_q := S \cap q$

Second query:

$q := [100, 200]$ and $s := 1$

Return e

Issue: Sample results of different queries can be correlated!

$S :=$ a set of real values

First query:

$q := [100, 200]$ and $s := 1$

Return a uniformly random element $e \in S_q := S \cap q$

Second query:

$q := [100, 200]$ and $s := 1$

Return e

Third query:

$q := [100, 200]$ and $s := 1$

Issue: Sample results of different queries can be correlated!

$S :=$ a set of real values

First query:

$q := [100, 200]$ and $s := 1$

Return a uniformly random element $e \in S_q := S \cap q$

Second query:

$q := [100, 200]$ and $s := 1$

Return e

Third query:

$q := [100, 200]$ and $s := 1$

Return e

Independent Query Sampling (PODS'14):
Sample results of different queries must be independent.

Independent Query Sampling (PODS'14):
Sample results of different queries must be independent.

$S :=$ a set of real values

Independent Query Sampling (PODS'14):
Sample results of different queries must be independent.

$S :=$ a set of real values

First query:

$q := [100, 200]$ and $s := 1$

Return a uniformly random element $e \in S_q := S \cap q$

Independent Query Sampling (PODS'14):
Sample results of different queries must be independent.

$S :=$ a set of real values

First query:

$q := [100, 200]$ and $s := 1$

Return a uniformly random element $e \in S_q := S \cap q$

Second query:

$q := [100, 200]$ and $s := 1$

Return a uniformly random element $e \in S_q := S \cap q$

Independent Query Sampling (PODS'14):
Sample results of different queries must be independent.

$S :=$ a set of real values

First query:

$q := [100, 200]$ and $s := 1$

Return a uniformly random element $e \in S_q := S \cap q$

Second query:

$q := [100, 200]$ and $s := 1$

Return a uniformly random element $e \in S_q := S \cap q$

Third query:

$q := [100, 200]$ and $s := 1$

Return a uniformly random element $e \in S_q := S \cap q$

Independent Query Sampling (PODS'14):
Sample results of different queries must be independent.

Independent Query Sampling (PODS'14):
Sample results of different queries must be independent.

- Given a predicate q and a sample size $s \geq 1$

Independent Query Sampling (PODS'14):
Sample results of different queries must be independent.

- Given a predicate q and a sample size $s \geq 1$
- Return a **size-s sample set** of S_q

Independent Query Sampling (PODS'14):
Sample results of different queries must be independent.

- Given a predicate q and a sample size $s \geq 1$
- Return a **size-s sample set** of S_q
  which is independent from the previous queries' outputs.

Independent Query Sampling (PODS'14):
Sample results of different queries must be independent.

- Given a predicate q and a sample size $s \geq 1$
- Return a **size-s sample set** of S_q
  which is independent from the previous queries' outputs.

Naive: Find S_q and then sample from it.

Independent Query Sampling (PODS'14):
Sample results of different queries must be independent.

- Given a predicate q and a sample size $s \geq 1$
- Return a **size-s sample set** of S_q
  which is independent from the previous queries' outputs.

Naive: Find S_q and then sample from it.
But: We want to do much faster when $s \ll |S_q|$.

Independent Query Sampling (PODS'14):
Sample results of different queries must be independent.

- Given a predicate q and a sample size $s \geq 1$
- Return a **size-s sample set** of S_q
  which is independent from the previous queries' outputs.

Naive: Find S_q and then sample from it.
But: We want to do much faster when $s \ll |S_q|$.

We need to organize the data in ways drastically different from the existing data structures designed to report S_q.

Independent Query Sampling (PODS'14):
Sample results of different queries must be independent.

- Given a predicate q and a sample size $s \geq 1$
- Return a **size-s sample set** of S_q
  which is independent from the previous queries' outputs.

Naive: Find S_q and then sample from it.
But: We want to do much faster when $s \ll |S_q|$.

We need to organize the data in ways drastically different from the existing data structures designed to report S_q.
$\Rightarrow$
IQS provides a fresh angle to revisit **every** reporting query in CS.

Independent Query Sampling (PODS'14):
Sample results of different queries must be independent.

Why IQS?

- Query estimation
- Approximation algorithms
- Fairness
- Representativeness and diversity
- ...

Many algorithmic techniques have been developed to support IQS:

- Alias method
- Tree sampling
- Alias augmentation
- Coverage
- Approximate coverage
- Random permutation
- I/O efficient
- ...

Most of the above techniques are friendly to practical implementation.

This talk aims to cover

- **Alias method**
- **Tree sampling**
- Alias augmentation
- **Coverage**
- Approximate coverage
- Random permutation
- I/O efficient
- ...

The three techniques already allow us to tackle many IQS problems with attractive guarantees.

This talk aims to cover

- **Alias method**
- **Tree sampling**
- Alias augmentation
- **Coverage**
- Approximate coverage
- Random permutation
- I/O efficient
- ...

The three techniques already allow us to tackle many IQS problems with attractive guarantees.

We will concentrate on **weighted sampling**.

Set Sampling

Set Sampling

$S :=$ a set of n elements

Set Sampling

$S :=$ a set of n elements
Each element $e \in S$ is associated with a **weight** $w(e) > 0$.

Set Sampling

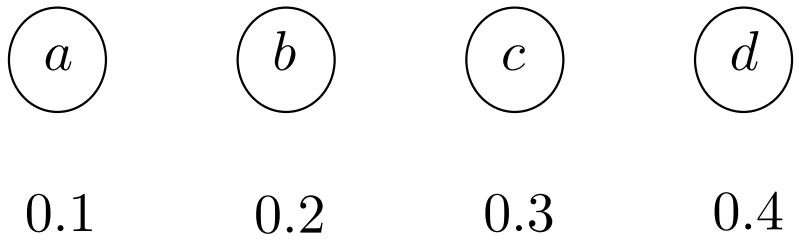
$S :=$ a set of n elements
Each element $e \in S$ is associated with a **weight** $w(e) > 0$.
A query extracts a **weighted sample**:

$$\boldsymbol{Pr}[e \text{ sampled}] = \frac{w(e)}{\sum_{e' \in S} w(e')}$$

Set Sampling

$S :=$ a set of n elements
Each element $e \in S$ is associated with a **weight** $w(e) > 0$.
A query extracts a **weighted sample**:

$$\boldsymbol{Pr}[e \text{ sampled}] = \frac{w(e)}{\sum_{e' \in S} w(e')}$$

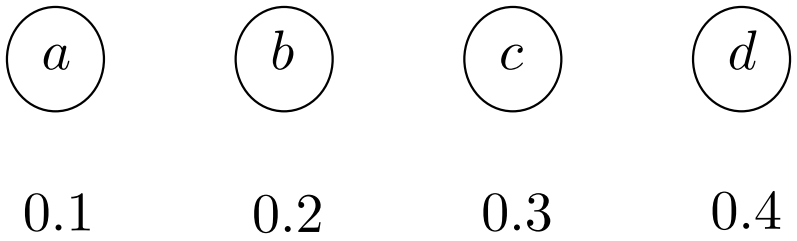


Set Sampling: Alias Method

Thm (Walker, 74): We can build a structure of $O(n)$ space in $O(n)$ time to answer each query in $O(1)$ time.

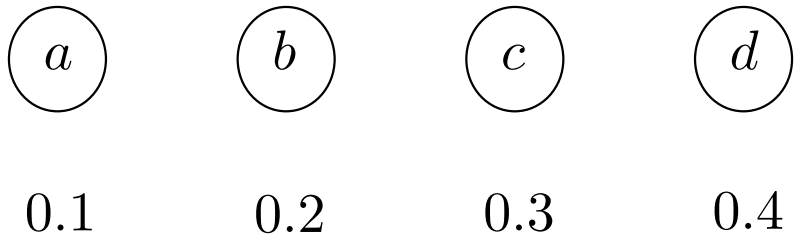
Set Sampling: Alias Method

Thm (Walker, 74): We can build a structure of $O(n)$ space in $O(n)$ time to answer each query in $O(1)$ time.



Set Sampling: Alias Method

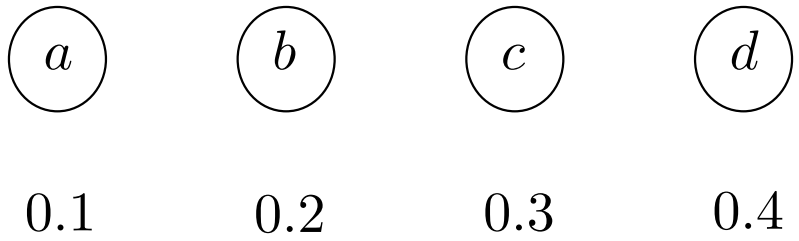
Thm (Walker, 74): We can build a structure of $O(n)$ space in $O(n)$ time to answer each query in $O(1)$ time.



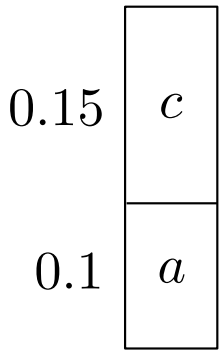
a = an element with weight ≤ 0.25, c = an element with weight ≥ 0.25

Set Sampling: Alias Method

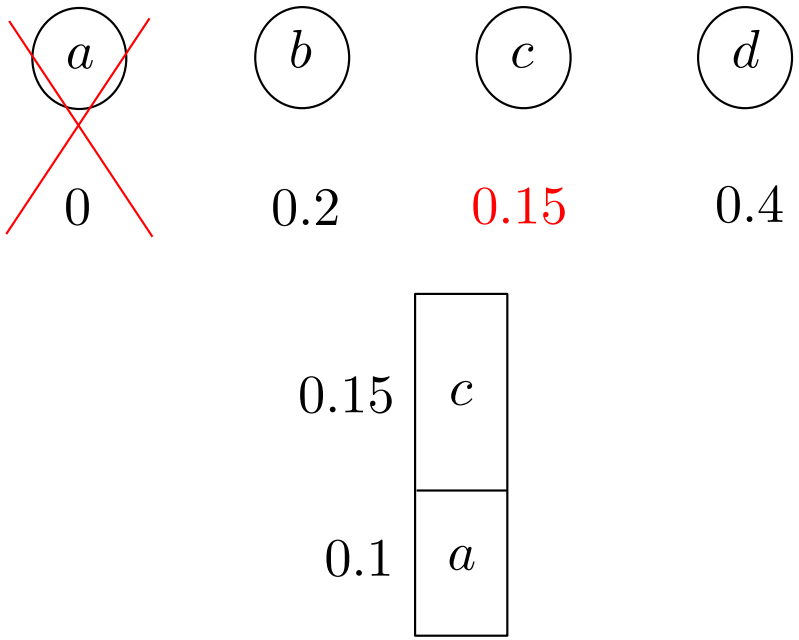
Thm (Walker, 74): We can build a structure of $O(n)$ space in $O(n)$ time to answer each query in $O(1)$ time.



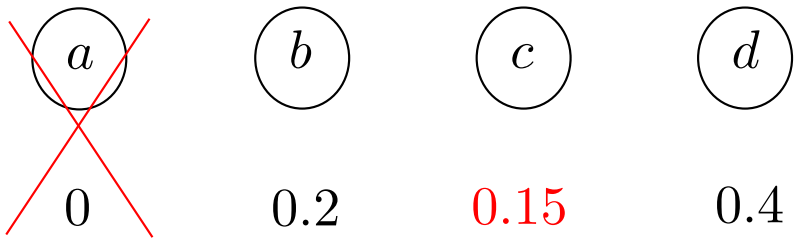
a = an element with weight ≤ 0.25, c = an element with weight ≥ 0.25



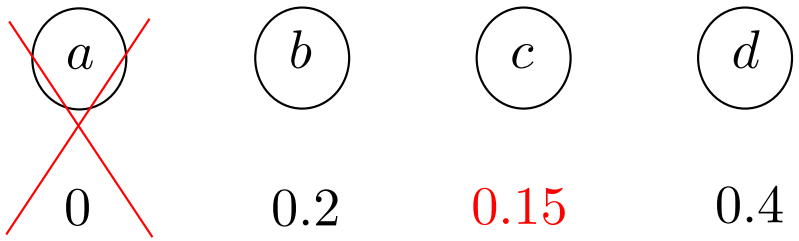
Set Sampling: Alias Method



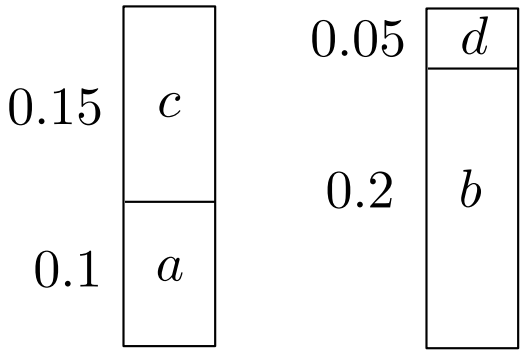
Set Sampling: Alias Method



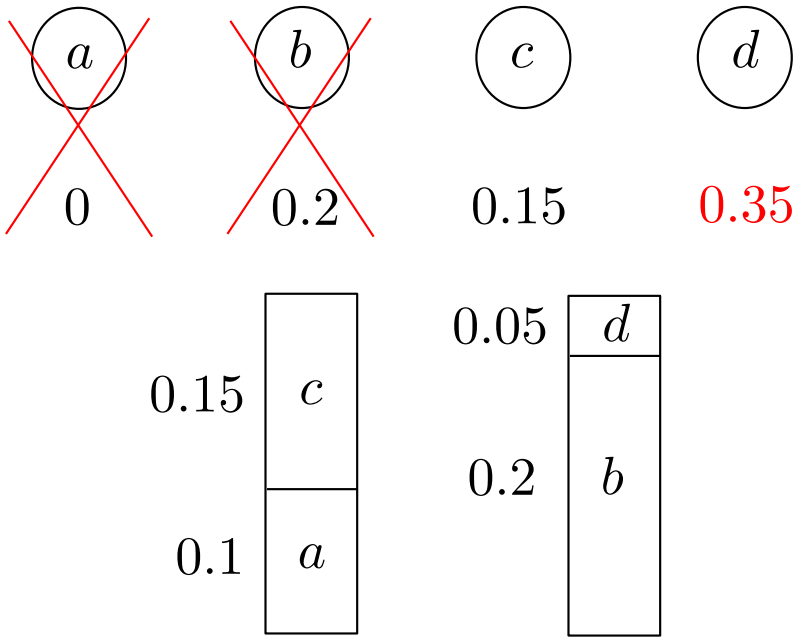
Set Sampling: Alias Method



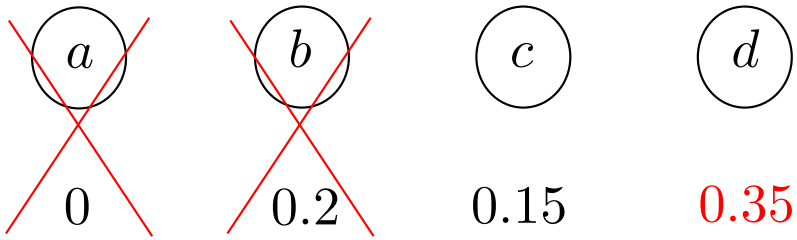
b = an element with weight ≤ 0.25, d = an element with weight ≥ 0.25



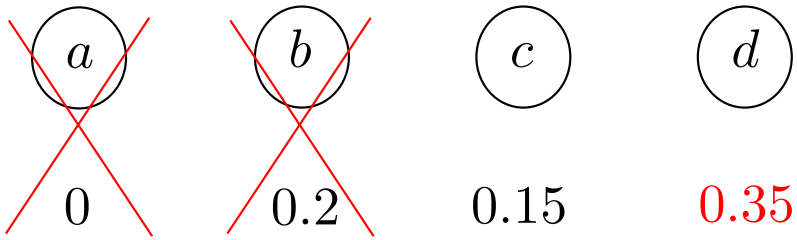
Set Sampling: Alias Method



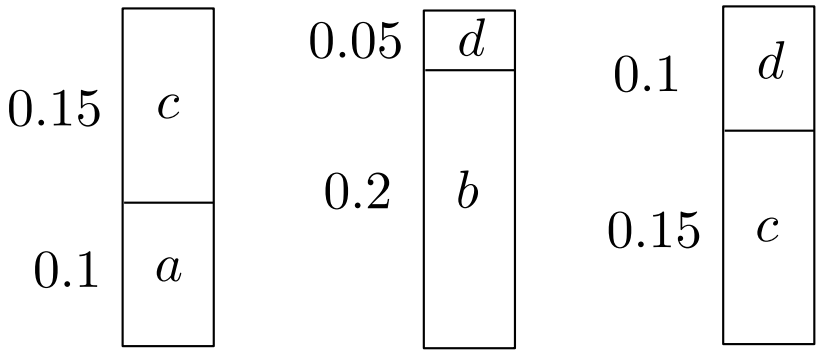
Set Sampling: Alias Method



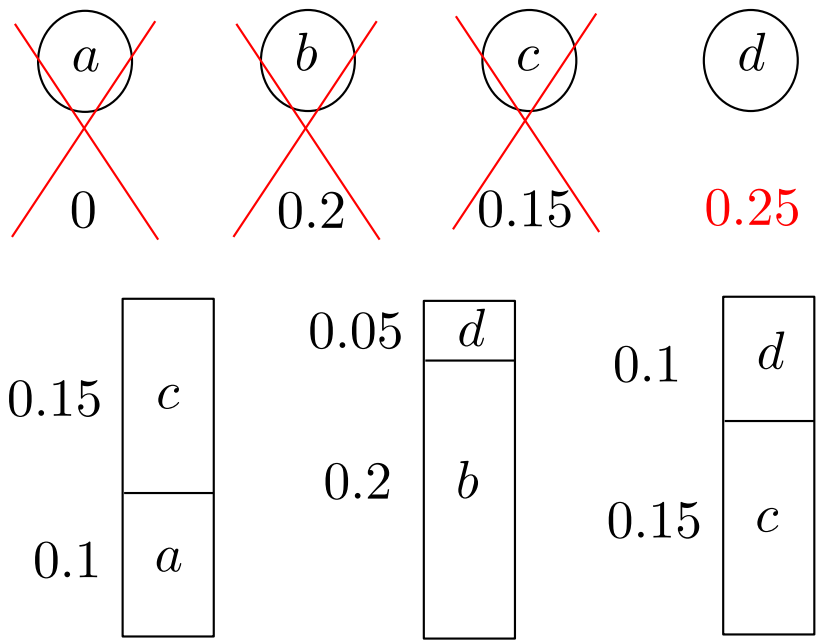
Set Sampling: Alias Method



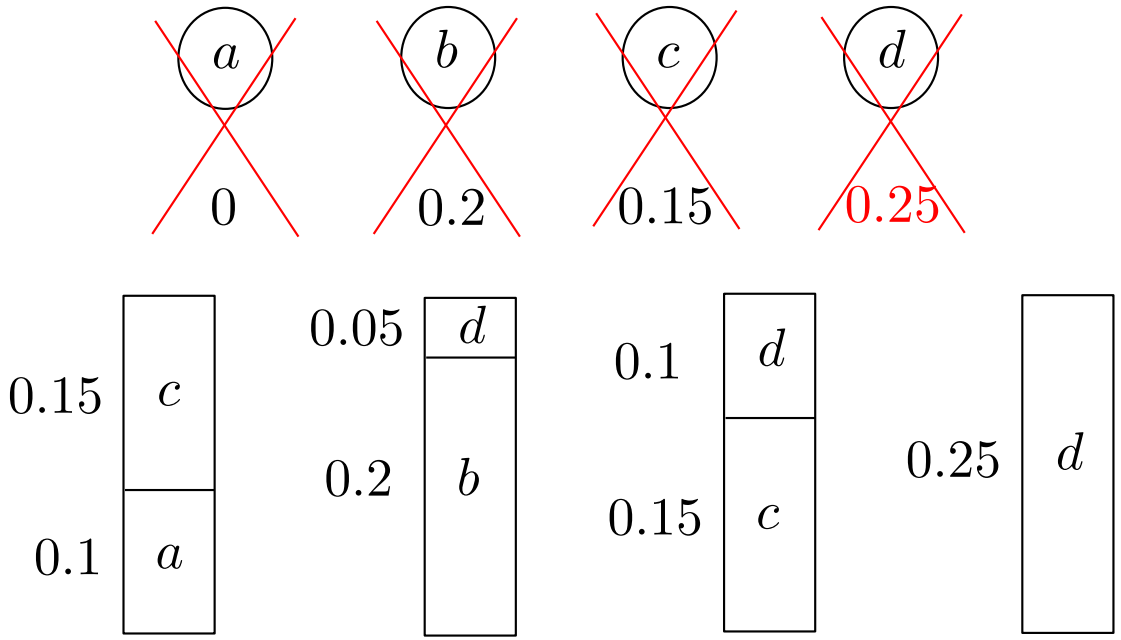
c = an element with weight ≤ 0.25, d = an element with weight ≥ 0.25



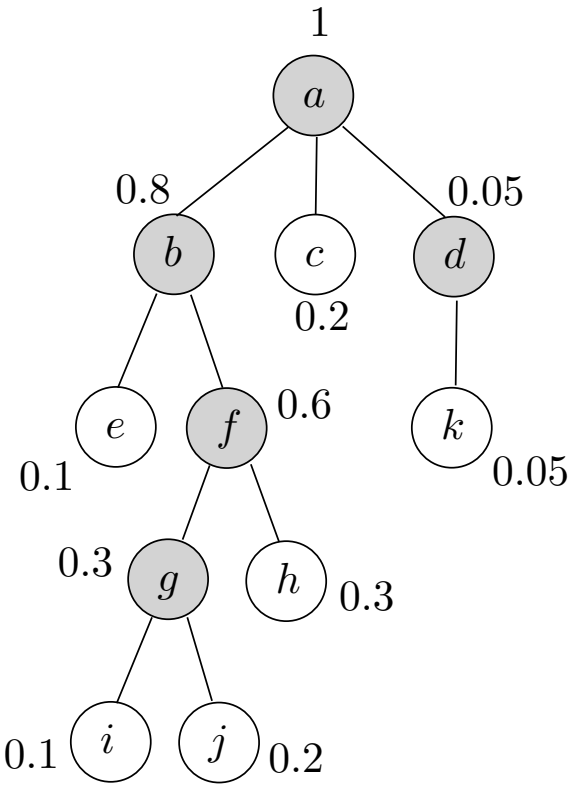
Set Sampling: Alias Method



Set Sampling: Alias Method

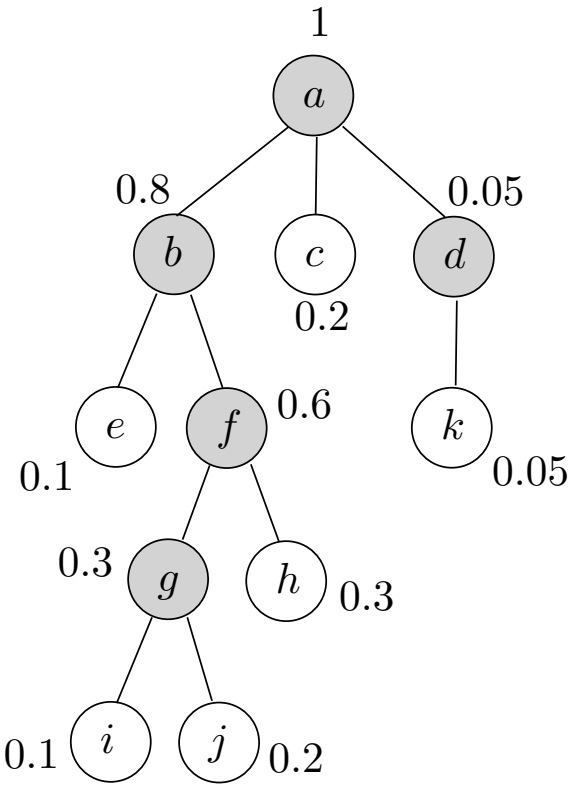


Tree Sampling



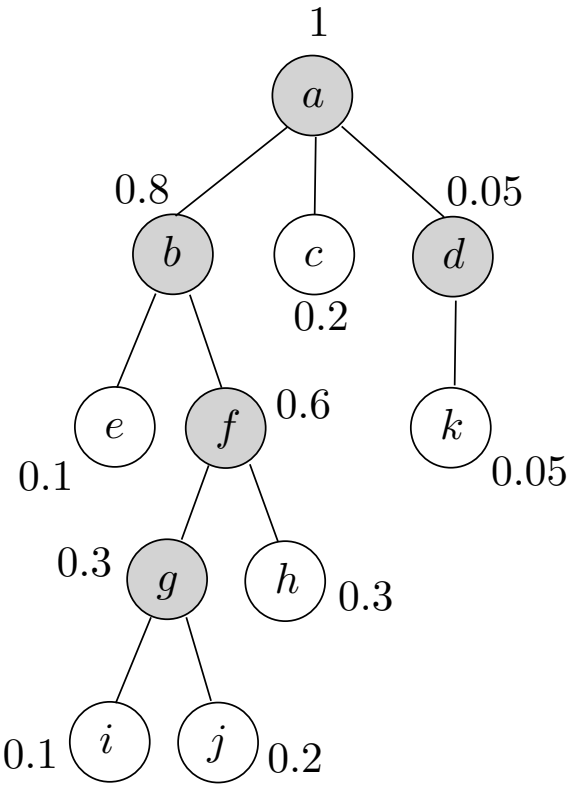
$\mathcal{T}$:= a tree of n nodes, where each leaf z carries a **weight** $w(z) > 0$.

Tree Sampling



$\mathcal{T}$:= a tree of n nodes, where each leaf z carries a **weight** $w(z) > 0$.
For an internal node u, $w(u)$:= total weight of its leaf descendants.

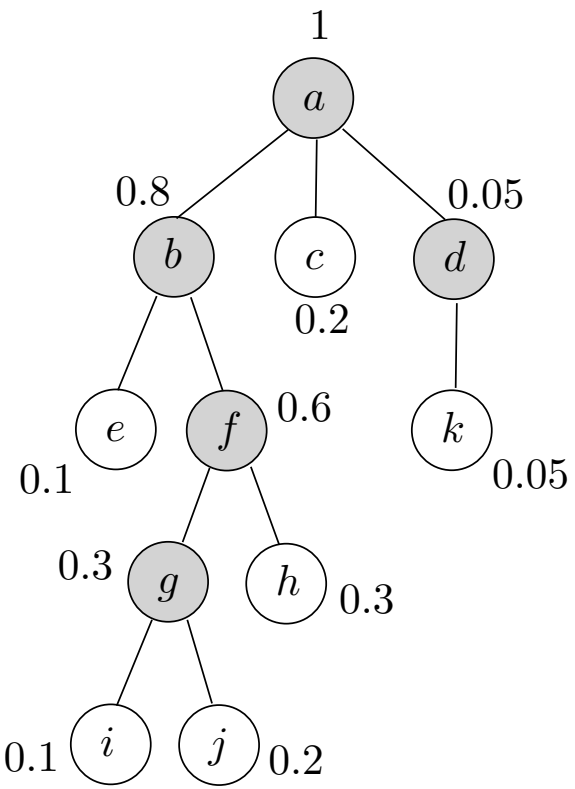
Tree Sampling



$\mathcal{T}$:= a tree of n nodes, where each leaf z carries a **weight** $w(z) > 0$.
For an internal node u, $w(u)$:= total weight of its leaf descendants.

> Given a node u, a query returns a **weighted sample** from the leaves of u.

Tree Sampling



Thm (Afshani and Phillips, 19): We can build a structure of $O(n)$ space to answer each query in $O(1)$ time.

Coverage

This technique can convert many data structures designed for (conventional) reporting to their IQS versions "for free".

Coverage

This technique can convert many data structures designed for (conventional) reporting to their IQS versions "for free".

S := input dataset; $\mathcal{T}$:= a tree on S

Coverage

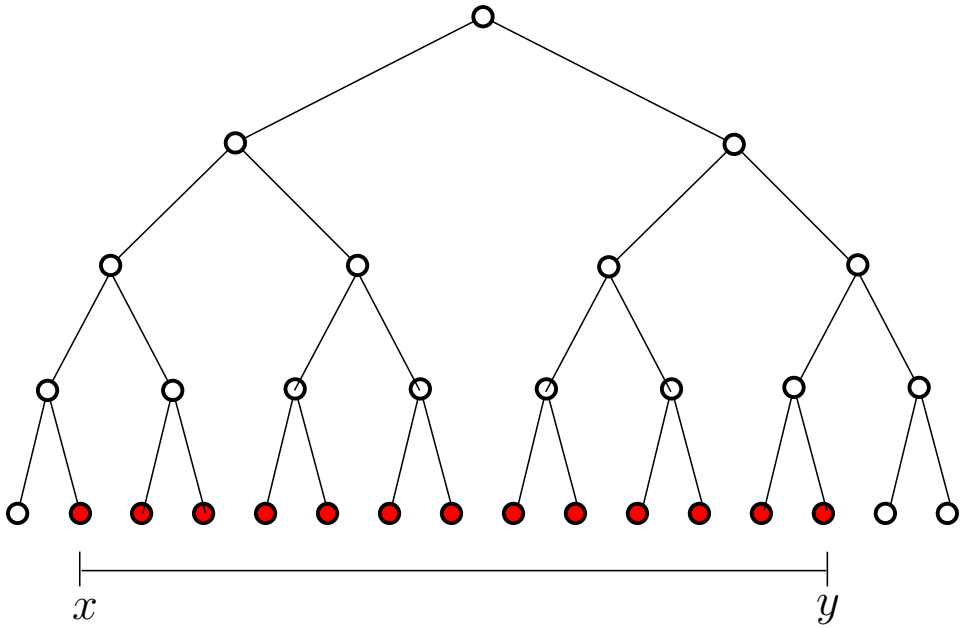
This technique can convert many data structures designed for (conventional) reporting to their IQS versions "for free".

$S :=$ input dataset; $\mathcal{T} :=$ a tree on S
For a predicate q, the result is a set S_q leaves in $\mathcal{T}$.

Coverage

This technique can convert many data structures designed for (conventional) reporting to their IQS versions "for free".

S := input dataset; $\mathcal{T}$:= a tree on S
For a predicate q, the result is a set S_q leaves in $\mathcal{T}$.



Coverage

$S :=$ input dataset; $\mathcal{T} :=$ a tree on S

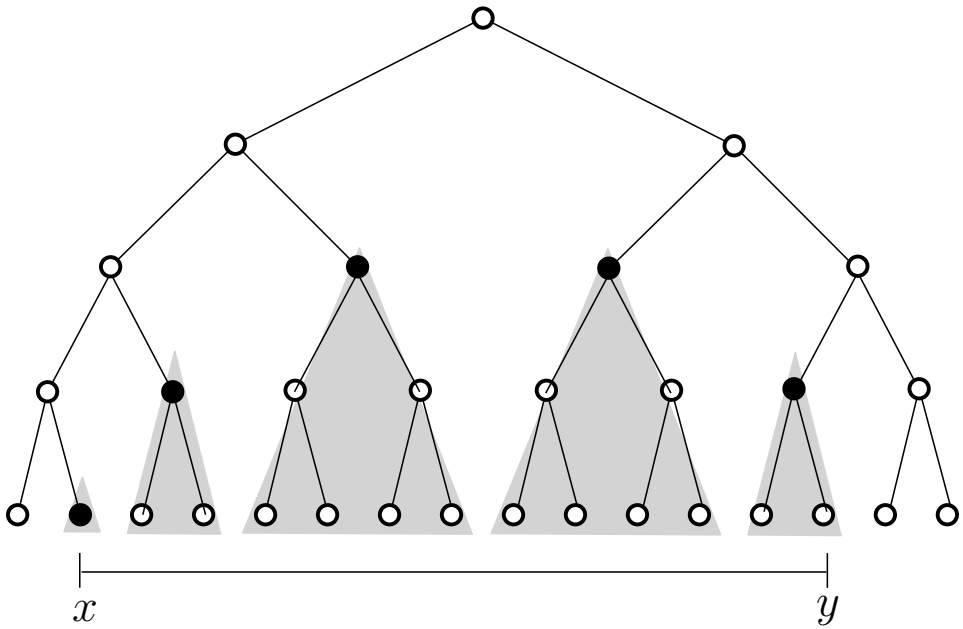
For a predicate q, the result is a set S_q leaves in $\mathcal{T}$.

Coverage

$S :=$ input dataset; $\mathcal{T} :=$ a tree on S
For a predicate q, the result is a set S_q leaves in $\mathcal{T}$.

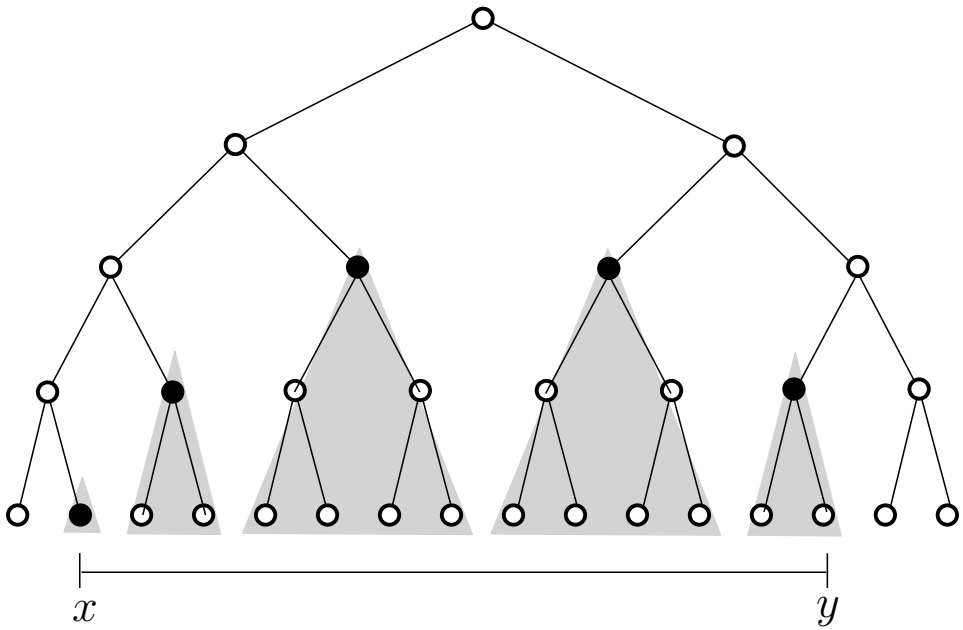
A **cover** $\mathcal{C}$ of $q :=$ a set of nodes in $\mathcal{T}$ s.t.

- the nodes in $\mathcal{C}$ have **disjoint** subtrees;
- the leaves in those subtrees constitute S_q.

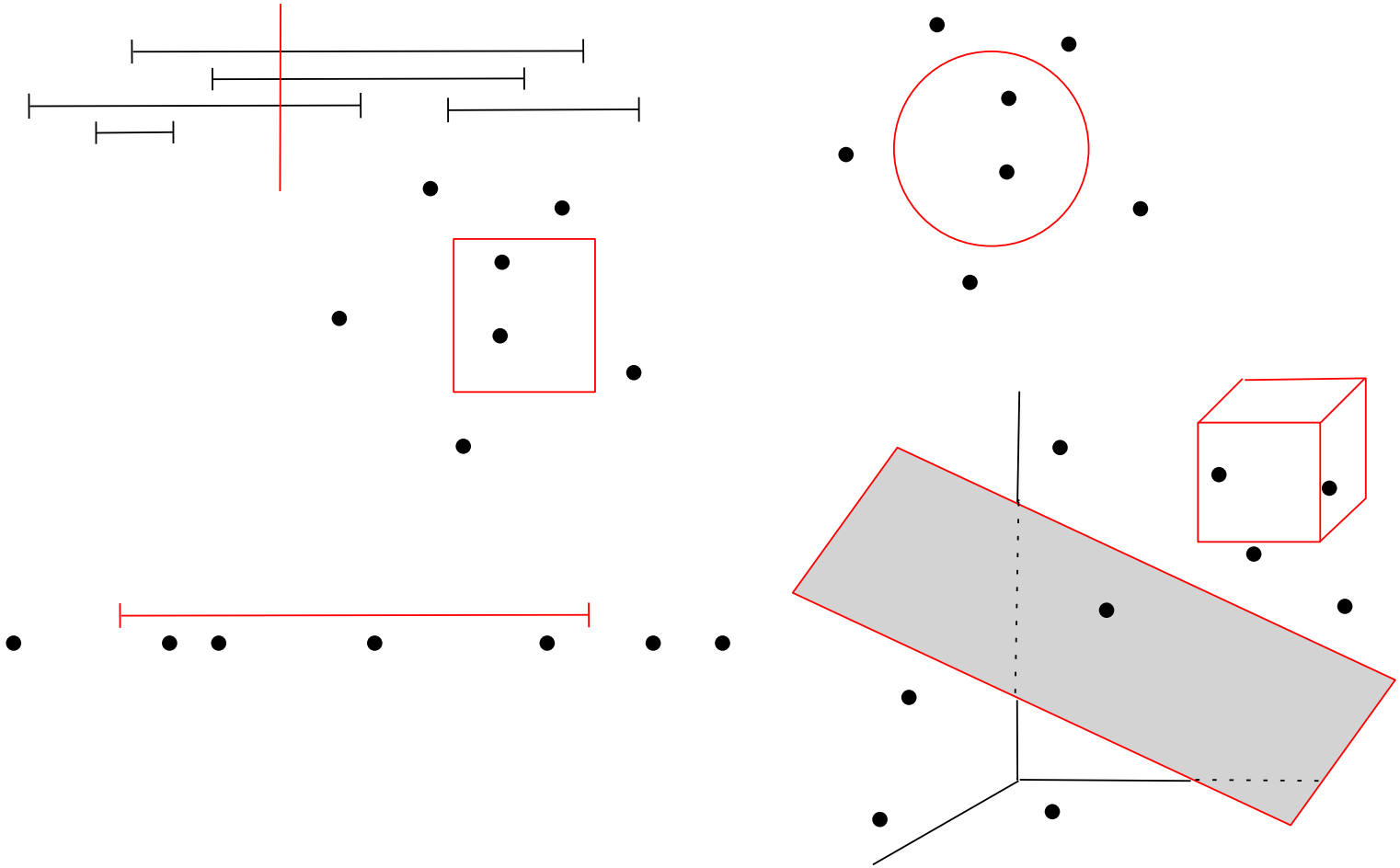


Coverage

Thm: An IQS query with predicate q and sample size s can be answered in $O(|\mathcal{C}| + s)$ time, plus the time to find $\mathcal{C}$.



Today, IQS structures with $\tilde{O}(n)$ space and $\tilde{O}(1) + O(s)$ query time are known for many problems.



Road ahead

- "Difficult" reporting queries
- Dynamization
- External memory
- Generic reductions
- Approximate IQS
- ...