Optimal Dynamic Parameterized Subset Sampling

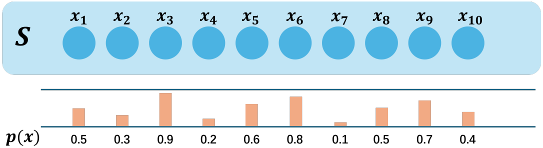
Junhao Gan[1], Seeun William Umboh[1,3], Hanzhi Wang[2], Anthony Wirth[4], **Zhuo Zhang**[1]

[1] The University of Melbourne, Australia
[2] BARC, University of Copenhagen, Denmark
[3] ARC Training Centre in Optimisation Technologies, Integrated Methodologies, and Applications (OPTIMA), Australia
[4] The University of Sydney, Australia

2025-6-23

Subset Sampling (SS)

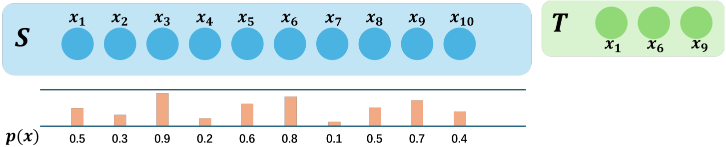
Given a set $S = \{x_1, \ldots, x_n\}$, where each item x has a fixed probability $p(x)$.



Subset Sampling (SS)

Given a set $S = \{x_1, \ldots, x_n\}$, where each item x has a fixed probability $p(x)$.

Goal: Sample a subset $T \subseteq S$ such that each item x is included in T independently with probability $p(x)$.



Subset Sampling (SS)

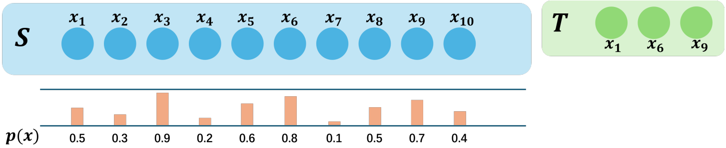
Given a set $S = \{x_1, \ldots, x_n\}$, where each item x has a fixed probability $p(x)$.

Goal: Sample a subset $T \subseteq S$ such that each item x is included in T independently with probability $p(x)$.

Optimal Query Time

The optimal query time is: $O(1+\mu)$ (in expectation), where $\mu = \sum_x p(x)$.

This bound is achievable with $O(n)$ preprocessing and $O(n)$ space.

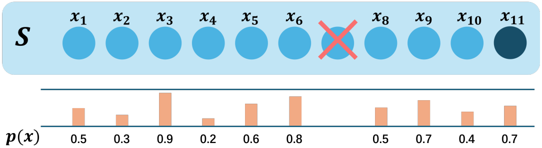


[Aggarwal–Vitter 1987, Bringmann–Friedrich 2020]

Dynamic Subset Sampling (DSS)

The item set S can be updated by:

- insertion of a new item x with fixed probability $p(x)$
- deletion of an existing item from S



Dynamic Subset Sampling (DSS)

The item set S can be updated by:

- insertions of a new item x with fixed probability $p(x)$
- deletions of an existing item from S

Dynamic Subset Sampling (DSS)

The item set S can be updated by:

- insertions of a new item x with fixed probability $p(x)$
- deletions of an existing item from S

Optimal Complexity

The optimal solution of DSS problem should achieve:

- $O(1+\mu)$ expected time per query
- $O(1)$ worst-case update time per insertion or deletion
- $O(n)$ space and $O(n)$ preprocessing time

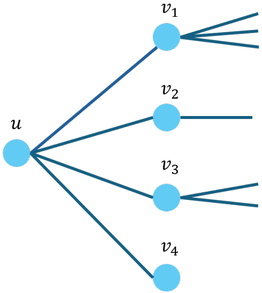
[Wang et al. 2023, Bhattacharya et al. 2023]

Motivating Example: Degree-based Random Walk

Consider the batch version of degree-based random walk on undirected dynamic graph.

Goal: Sample a random subset $T \subseteq N(u)$ such that each $v \in N(u)$ is selected independently with probability

$$p(v) = \frac{\deg(v)}{\sum_{v' \in N(u)} \deg(v')}$$

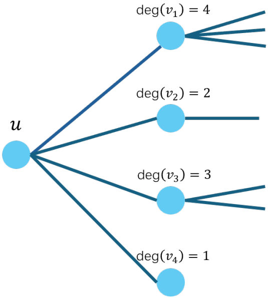


Motivating Example: Degree-based Random Walk

Consider the batch version of degree-based random walk on undirected dynamic graph.

Goal: Sample a random subset $T \subseteq N(u)$ such that each $v \in N(u)$ is selected independently with probability

$$p(v) = \frac{\deg(v)}{\sum_{v' \in N(u)} \deg(v')}$$

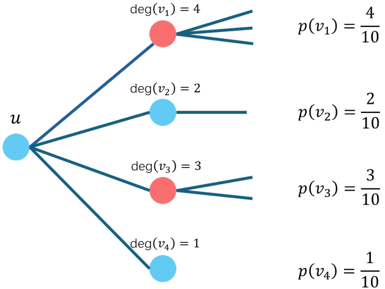


Motivating Example: Degree-based Random Walk

Consider the batch version of degree-based random walk on undirected dynamic graph.

Goal: Sample a random subset $T \subseteq N(u)$ such that each $v \in N(u)$ is selected independently with probability

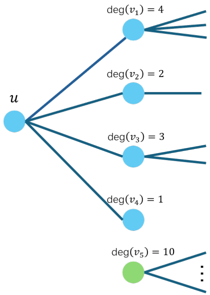
$$p(v) = \frac{\deg(v)}{\sum_{v' \in N(u)} \deg(v')}$$



Motivating Example: Weighted Subset Sampling

Challenge: When $N(u)$ is updated (e.g., inserting a new high-degree node), all probabilities $p(v)$ change.

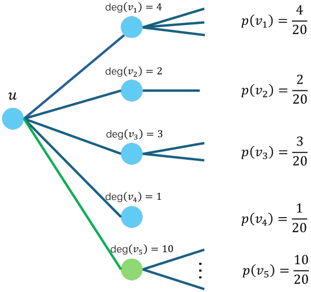
DSS cannot handle this problem trivially: instead, it fixes $p(v)$ at insertion time.



Motivating Example: Weighted Subset Sampling

Challenge: When $N(u)$ is updated (e.g., inserting a new high-degree node), all probabilities $p(v)$ change.

DSS cannot handle this problem trivially: instead, it fixes $p(v)$ at insertion time.



Dynamic Parameterized Subset Sampling in the Word RAM Model

Parameterized Subset Sampling (PSS)

Given a dynamic set S of n items, where each item $x \in S$ has a non-negative integer weight $w(x)$.

Goal: For any pair of non-negative rational parameters (α, β), return a random subset $T \subseteq S$ such that each item $x \in S$ is included independently with probability

$$p_x(\alpha, \beta) = \min\left\{1, \frac{w(x)}{W_S(\alpha, \beta)}\right\}, \quad \text{where } W_S(\alpha, \beta) = \alpha \cdot \sum_{x \in S} w(x) + \beta$$

Parameterized Subset Sampling (PSS)

Given a dynamic set S of n items, where each item $x \in S$ has a non-negative integer weight $w(x)$.

Goal: For any pair of non-negative rational parameters (α, β), return a random subset $T \subseteq S$ such that each item $x \in S$ is included independently with probability

$$p_x(\alpha, \beta) = \min\left\{1, \frac{w(x)}{W_S(\alpha, \beta)}\right\}, \quad \text{where } W_S(\alpha, \beta) = \alpha \cdot \sum_{x \in S} w(x) + \beta$$

Interpreting Parameters:

- If $\alpha = 0$, β is a constant: recovers DSS problem
- If $\alpha = 1, \beta = 0$: recovers score-based subset sampling problem

  user can tune α to control expected sample size

The Word RAM Model

We adopt the standard Word RAM model with word length d bits, where

$$d \in \Omega(\log(n_{\max} \cdot w_{\max}))$$

Each atomic operation on $O(1)$-word integers can be performed in $O(1)$ time:

- **Arithmetic:** $+, -, \times$, division with rounding
- **Bit operations:** e.g. find the index of the highest non-zero bit
- **Randomness:** generate a uniformly random word of d bits

Our Results

We can achieve the following optimal complexity

Theorem 1. For the DPSS problem on a set S of n items, there exists an algorithm which achieves the following bounds in the Word RAM model:

Pre-processing Time: $O(n)$ worst-case;

Query Time: $O(1 + \mu)$ in expectation;

Update Time: $O(1)$ worst-case;

Space Consumption: $O(n)$ worst-case at all times.

Hardness of DPSS with Float Weights

Suppose there exists an algorithm for **deletion-only DPSS with float weights** that achieves:

- **Preprocessing time:** $O(n)$
- **Query time:** $O(1+\mu)$ expected
- **Update time:** $O(1)$ worst-case

Then: **Integer Sorting** of n integers with $d \in \Omega(\log n)$ bits can be solved in $O(n)$ expected time, which is still an open problem*.

$*$See [Belazzougui et al., 2014] for related work on integer sorting.

Hardness of DPSS with Float Weights

Suppose there exists an algorithm for **deletion-only DPSS with float weights** that achieves:

- **Preprocessing time:** $O(n)$
- **Query time:** $O(1 + \mu)$ expected
- **Update time:** $O(1)$ worst-case

Then: **Integer Sorting** of n integers with $d \in \Omega(\log n)$ bits can be solved in $O(n)$ expected time, which is still an open problem*.

This suggests that solving **float-weight DPSS optimally** is likely hard.

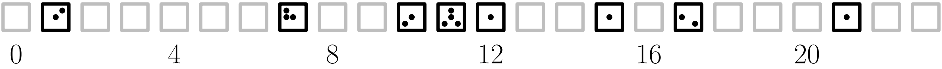
*See [Belazzougui et al., 2014] for related work on integer sorting.

Our Algorithm

Bucketing-Based Algorithm: A Warm-Up

We organize items into **power-of-two buckets**:

- Bucket $B(i)$ contains items with weights in $[2^i, 2^{i+1})$



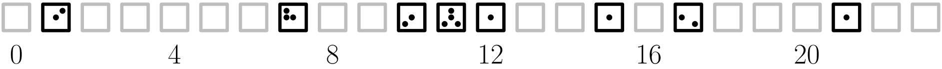
Bucketing-Based Algorithm: A Warm-Up

We organize items into **power-of-two buckets**:

- Bucket $B(i)$ contains items with weights in $[2^i, 2^{i+1})$

A simple algorithm works as:

- For each non-empty bucket $B(i)$:
  - Sample potential items using upper-bound probability $p'_x = \min\left\{1, \frac{2^{i+1}}{W_S(\alpha,\beta)}\right\}$
  - Accept each potential item x with probability $\frac{p_x}{p'_x}$



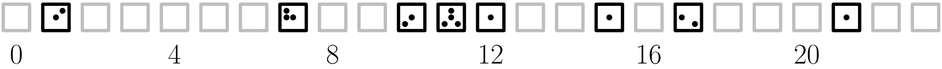
Bucketing-Based Algorithm: A Warm-Up

We organize items into **power-of-two buckets**:

- Bucket $B(i)$ contains items with weights in $[2^i, 2^{i+1})$

A simple algorithm works as:

- For each non-empty bucket $B(i)$:
  - Sample potential items using upper-bound probability $p'_x = \min\left\{1, \frac{2^{i+1}}{W_S(\alpha,\beta)}\right\}$
    (done efficiently via generating geometric random variable)
  - Accept each potential item x with probability $\frac{p_x}{p'_x}$



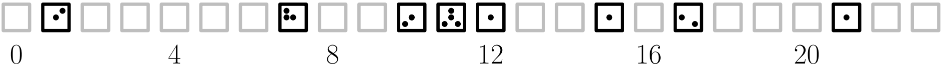
Bucketing-Based Algorithm: A Warm-Up

We organize items into **power-of-two buckets**:

- Bucket $B(i)$ contains items with weights in $[2^i, 2^{i+1})$

A simple algorithm works as:

- For each non-empty bucket $B(i)$:
  - Sample potential items using upper-bound probability
    $p'_x = \min\left\{1, \frac{2^{i+1}}{W_S(\alpha,\beta)}\right\}$
    (done efficiently via generating geometric random variable)
  - Accept each potential item x with probability $\frac{p_x}{p'_x}$
    (guaranteed $\geq 1/2$)



Bucketing-Based Algorithm: A Warm-Up

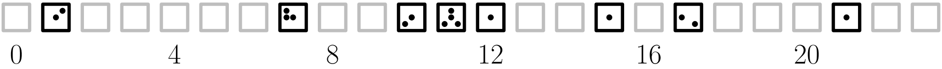
We organize items into **power-of-two buckets**:

- Bucket $B(i)$ contains items with weights in $[2^i, 2^{i+1})$

A simple algorithm works as:

- For each non-empty bucket $B(i)$:
  - Sample potential items using upper-bound probability $p'_x = \min\left\{1, \frac{2^{i+1}}{W_S(\alpha,\beta)}\right\}$ (done efficiently via generating geometric random variable)
  - Accept each potential item x with probability $\frac{p_x}{p'_x}$ (guaranteed $\geq 1/2$)

Query Time: $O(b + \mu)$ expected, where b is the number of non-empty buckets.



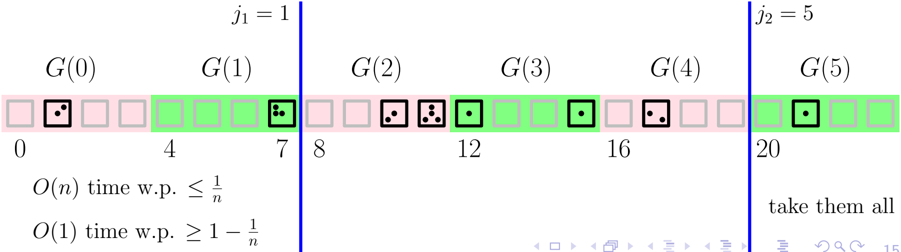
Towards Optimal Query Time

Challenge: How can we avoid touching all the buckets?

Towards Optimal Query Time

Challenge: How can we avoid touching all the buckets?

Our solution: Partition the buckets into groups!



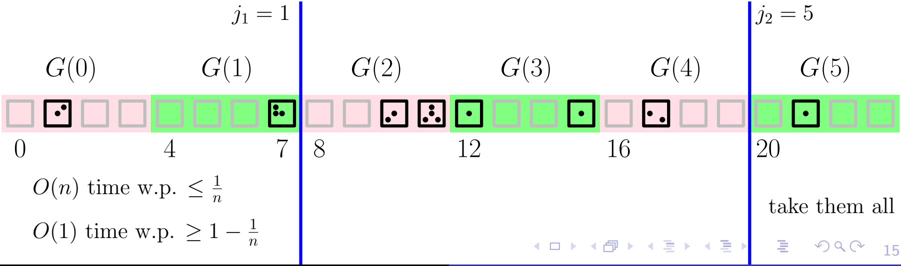
Towards Optimal Query Time

Challenge: How can we avoid touching all the buckets?

Our solution: Partition the buckets into groups!

- **Insignificant Groups:** all items have sampling probability $< \frac{1}{n^2}$

  $\Rightarrow$ This is a easy case, since probability that at least one item is sampled $\leq 1/n$



Towards Optimal Query Time

Challenge: How can we avoid touching all the buckets?

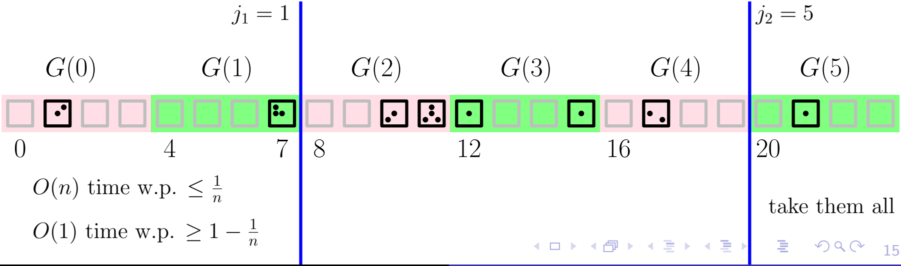
Our solution: Partition the buckets into groups!

- **Insignificant Groups:** all items have sampling probability $< \frac{1}{n^2}$

  $\Rightarrow$ This is a easy case, since probability that at least one item is sampled $\leq 1/n$

- **Certain Groups:** all items have sampling probability ≥ 1

  $\Rightarrow$ Output all items directly.



Towards Optimal Query Time

Challenge: How can we avoid touching all the buckets?

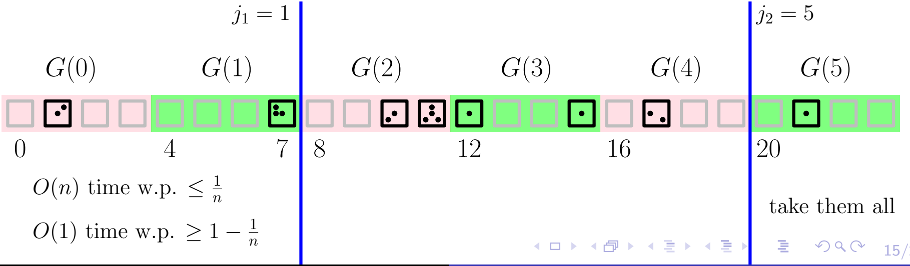
Our solution: Partition the buckets into groups!

- **Insignificant Groups:** all items have sampling probability $< \frac{1}{n^2}$

  $\Rightarrow$ This is a easy case, since probability that at least one item is sampled $\leq 1/n$
- **Certain Groups:** all items have sampling probability ≥ 1

  $\Rightarrow$ Output all items directly.
- **Significant Groups:** all of the other groups

  $\Rightarrow$There are at most 3 significant groups.



Handling Significant Groups

- Step 1: Find the potential buckets, i.e., those containing at least one potential item.

Handling Significant Groups

- Step 1: Find the potential buckets, i.e., those containing at least one potential item.
  - This is another subset sampling problem! Recursion!

Handling Significant Groups

- Step 1: Find the potential buckets, i.e., those containing at least one potential item.
  - This is another subset sampling problem! Recursion!
  - After three times recursion, the problem size is $O(\log\log\log n)$.

Handling Significant Groups

- Step 1: Find the potential buckets, i.e., those containing at least one potential item.
  - This is another subset sampling problem! Recursion!
  - After three times recursion, the problem size is $O(\log\log\log n)$.
  - Small enough to be solved in a pre-computed look-up table.

Handling Significant Groups

- Step 1: Find the potential buckets, i.e., those containing at least one potential item.
  - This is another subset sampling problem! Recursion!
  - After three times recursion, the problem size is $O(\log\log\log n)$.
  - Small enough to be solved in a pre-computed look-up table.
- Step 2: Sample from the potential buckets.

Handling Significant Groups

[Find First Potential Item]: How to efficiently find the index k of the first potential item in a bucket $B(i)$ conditioned on the fact that $B(i)$ contains at least one?

Handling Significant Groups

[Find First Potential Item]: How to efficiently find the index k of the first potential item in a bucket $B(i)$ conditioned on the fact that $B(i)$ contains at least one?

This is a **conditional probability problem**:

- Each item is sampled independently with probability p
- Conditioned on at least one sample occurs

Key Idea: Use the Truncated Geometric Distribution T-Geo(p, n)

$$\Pr[\text{T-Geo}(p, n) = i] = \frac{p(1-p)^{i-1}}{1-(1-p)^n} \quad \text{for } i \in \{1, \ldots, n\}$$

Background: Truncated Geometric in Word RAM

Problem: How to generate T-Geo(p, n) variables in the Word RAM model?

Background: Truncated Geometric in Word RAM

Problem: How to generate T-Geo(p, n) variables in the Word RAM model?

Naive implementation idea: Use the Inverse Transform Sampling

$$\left\lfloor \frac{\log(1 - \text{rand}(0,1) \cdot (1 - (1-p)^n))}{\log(1-p)} \right\rfloor + 1$$

to simulate T-Geo(p).

Background: Truncated Geometric in Word RAM

Problem: How to generate T-Geo(p, n) variables in the Word RAM model?

Naive implementation idea: Use the Inverse Transform Sampling

$$\left\lfloor \frac{\log(1 - \text{rand}(0,1) \cdot (1 - (1-p)^n))}{\log(1-p)} \right\rfloor + 1$$

to simulate T-Geo(p).

But this relies on floating-point logarithm and arbitrary precision rounding—**not supported** in the Word RAM model.

Background: Truncated Geometric in Word RAM

Problem: How to generate T-Geo(p, n) variables in the Word RAM model?

Naive implementation idea: Use the Inverse Transform Sampling

$$\left\lfloor \frac{\log(1 - \mathrm{rand}(0,1) \cdot (1 - (1-p)^n))}{\log(1-p)} \right\rfloor + 1$$

to simulate T-Geo(p).

But this relies on floating-point logarithm and arbitrary precision rounding—**not supported** in the Word RAM model.

Prior Work: Bringmann and Friedrich (SODA'13) designed $O(1)$ time Word RAM algorithms for:

- B-Geo(p, n):

$$\Pr[\text{B-Geo}(p,n) = i] = \begin{cases} p(1-p)^{i-1} & i \in \{1, \cdots, n-1\}\,; \\ (1-p)^{n-1} & i = n\,. \end{cases}$$

About Random Variates Generation

Let p be a rational number in $(0, 1)$ which can be represented by a $O(1)$-word integer nominator and a $O(1)$-word integer denominator.

Our algorithm used the following five types of random variates:

- $\mathrm{Ber}(p)$ (by Bringmann and Friedrich)
- $\mathrm{Ber}(\frac{1-(1-p)^n}{p \cdot n})$ (new by us)
- $\mathrm{Ber}(\frac{\frac{1}{2} \cdot p \cdot n}{1-(1-p)^n})$ (new by us)
- $\text{B-Geo}(p, n)$ (by Bringmann and Friedrich)
- $\text{T-Geo}(p, n)$ (new by us)

Each of the above random variates can be generated in $O(1)$ expected time with $O(n)$ worst-case space.

Conclusions

Conclusions

We formulated the DPSS problem.

We proposed an exact and optimal algorithm for DPSS in the Word RAM model.

We gave efficient and exact generation algorithms for a number of random vairates in Word RAM model.

We showed a hardness result on deletion-only DPSS with float weights.

Conclusions

We formulated the DPSS problem.

We proposed an exact and optimal algorithm for DPSS in the Word RAM model.

We gave efficient and exact generation algorithms for a number of random vairates in Word RAM model.

We showed a hardness result on deletion-only DPSS with float weights.

Thank you! Questions are welcomed.

A Reduction

Consider a set of N integers $I = \{a_1, \ldots, a_N\}$, each of which is represented by one word of d bits.

The set I can be sorted in descending order by the following algorithm:

- for each integer $a_i \in I$, create an item x_i with weight $w(x_i) = 2^{a_i}$, represented by a float number;
- initialize S to be the set of all these N items;
- initialize an empty linked list, R, of the integers in I, which is maintained to be sorted, in descending order, by the Insertion Sort algorithm;

A Reduction

- initialize a deletion-only *DPSS-ALG* on S;
- while S is not empty, perform the following:
  - *repeatedly* invoke *DPSS-ALG* on S to perform a PSS query with parameters $(1, 0)$ until the sampling result $T \neq \emptyset$;
  - let x^* be the item in T with the *largest* weight $w(x^*) = 2^{a^*}$;
  - invoke *DPSS-ALG* to delete x^* from S;
  - invoke Insertion Sort to insert the weight exponent, a^*, to R;
- return R as the sorted list of all the integers in I;