IPPS Stream Reservoir Sampling

- Each arriving item:
  - Provisionally include item in reservoir
  - If $m+1$ items, discard 1 item randomly (same as include $m$ items randomly)
  - Choose inclusion probabilities to be previous IPPS
    - Calculate threshold $z$ to include $m$ items on average: $z$ solves $\sum_i p_z(x_i) = m$
    - Discard item $i$ with probability $q_i = 1 - p_z(x_i)$
  - Adjust $m$ surviving $x_i$ with Horvitz-Thompson $x'_i = x_i / p_i = \max\{x_i, z\}$

Example: $m=9$

$\sum_{i}^{10} x_i = 10$

Recalculate threshold $z$:

Recalculate discard probs:

$x'_i = \max\{x_i, z\}$

Adjust weights:

$1 - p_z(x_i) = 1 - \min\{1, x_i / z\}$
Computation in IPPS Stream Sampling

• “Calculate threshold $z$ to include $m$ average items: $z$ solves $\sum_i p_z(x_i) = m$”?

• Weight order: $x_{(1)} \leq x_{(2)} \leq \ldots \leq x_{(n)}$. Any $z$: small items: $x_{(i)} \leq z$; Large $x_{(i)} > z$

• If $x_{(t)}$ is a small item, then:
  
  $m = \sum_i p_z(x_{(i)}) = \sum_i \min\{1, x_{(i)}/z\}$  (implicit definition of $z$, using form of $p_z$)

  $\leq \sum_{i=t}^n x_{(i)}/z + (m+1-t)$  ($p_z(x_{(i)}) \leq 1$ for the $m+1-t$ terms $(i) > t$)

  $\leq \sum_{i=t}^n x_{(i)}/x_{(t)} + (m+1-t)$  ($x_{(t)}$ small and hence $\leq z$)

• In other words: $\sum_{i=t}^n x_{(i)}/x_{(t)} \geq t-1$

• Largest possible index $t$ for small item:
  
  $t^* = \max\{t: \sum_{i=t}^n x_{(i)}/x_{(t)} \geq t-1\}$

• Then find $z$ from $\sum_{i=t^*}^n x_{(i)}/z = t^*-1$ (why?)

• How to find $t^*$?
How to find t*?

- \( t^* = \max\{ t: \sum_{i\leq t} x(i) / x(t) \geq t-1 \} \)

- **Exercise:**
  - show \( g(t) = \sum_{i\leq t} x(i) / x(t) - t - 1 \) is nonincreasing in \( t \).

- Show that this makes \( t^* \) easier to find
Monotonic Functions

• Searching for changepoints
  – f is binary function on \{1, 2, …, n\}: f(t) is either 0 or 1
  – f is monotonic: for some \( t^* \), \( f(t) = 1 \) if and only if \( t > t^* \)
  – Example:
    \[
    f(t) = 0 \text{ if } \sum_{i \leq t} x(i) / x(t) \geq t-1; \quad f(t) = 1 \text{ if } \sum_{i \leq t} x(i) / x(t) < t-1;
    \]
    \[
    t^* = \max\{t: f(t) = 0\} = \max\{ t: \sum_{i \leq t} x(i) / x(t) \geq t-1\}
    \]

• Task
  – Find changepoint \( t^* \)

• Simple approach:
  – inspect \( f(t) \) for \( t = 1, 2, \ldots, t^* = \text{last } t \text{ for which } f(t) = 0 \)
  – computational cost: \( O(n) \): might have to inspect all \( n \)
Binary Search for Changepoint

- **Initialize:**
  - Set t in center

- **Iterate (until can move t no further right)**
  - If \( f(t) = 1 \)
    - we know \( t^* < t \)
    - restrict attention to lower half subinterval, set t to its center
  - If \( f(t) = 0 \)
    - we know \( t^* \geq t \)
    - restrict attention to upper half subinterval, set t to its center

- \( t^* = t \)
- If h is number of halvings then \( 2^h \sim n \)
- \#iterations = number of halvings \( h = O(\log n) < O(n) \)
Need to generalize binary search

• Domain may be point subset of real numbers
  – E.g. set of weight $x_i$, not simply \{1,2,…,n\}
• Domain can change with time
  – Database insertions and deletions
  – For a given $f$, this can alter changepoint: need to locate again
• Need general way
  – To store/retrieve data points
  – That enables finding function changepoints
  – Abstracts the idea of binary search from any detailed setting
  – Is efficient
Introducing binary search trees

- Data structure to store and retrieve points in ordered set
  - e.g. numbers with order of "<"
- Points do not need to be added in any particular order
  - No presorting needed
- Tree like structure is very efficient
  - Stores \( n = 2^h - 1 \) items if depth \( h \)
  - Computational cost \( O(\log n) \) to retrieve any item
  - We will use to locate changepoint of binary monotonic function
Storing in a Binary Search Tree

• Storing an ordered set, e.g., \{17, 4, 6, 23, 36, 4\} with > order
  – Store first element at root” 17
  – Pass further elements down tree
    • To left child if element ≤ child
    • To right child if element > child

• Tree is depth \(O(\log m)\) for \(m\) items
• Computational complexity to add, delete or retrieve item is \(O(\log m)\)
• Binary search easy: go left/right until found, \(O(\log m)\) steps
• Need to **rebalance**: maintain bounded depth (i.e. approx. symmetry)
Search on a Binary Tree

• Search for some node (and whatever information attached)
  – say node 9
• Start at root
• Iterate until found:
  – Branch left if 9 ≤ node
  – Branch right if 9 > node

Found node 9!
Monotone Functions and Binary Search Tree

- Monotone function $f$ on nodes:
  - $f(\text{node}_1) \leq f(\text{node}_2)$ if and only if $\text{node}_1 < \text{node}_2$
  - Example: $f(\text{node}) = \text{weight stored at node}$

- Monotone binary function
  - Takes values 0 or 1
  - Seek node* = largest node with $f(\text{node}) = 0$
  - Example: largest node with weight $\leq 10$
  - Happens to be 9: not known at start
  - $f(\text{node}) = 0$ if weight $\leq 10$, 1 otherwise

- Start at root and iterate until can’t move further right:
  - if( $f(\text{node}) = 1$ ) {
    node $> \text{node}^*$: so branch left
  }
  - if( $f(\text{node}) = 0$ ) {
    node $\leq \text{node}^*$: so branch right
  }
  - $\text{node} = \text{node}^*$

---

Diagram:

- $f(17) = 1$
- $f(4) = 0$
- $f(6) = 0$
- $f(9) = 0$
- $f(23) = 0$
- $f(36) = 0$
- $f(25) = 0$

(can’t move further right: $\text{node}^* = 9$)
Back to IPPS: maintaining partial sums $\sum_{i \leq t} x_i$

- Insert: each node $x_i$ maintains partial sums $S_i$ of weights $x_j \leq x_i$ that traverse it.
- Recovery: from $x_i$ add up partial sums $S_j$ from all nodes $x_j \leq x_i$ on path back to root.
- Weight of items $\leq 25$: $25+23+17+4+6+4+9$
- Weight of items $\leq 9$: $9+6+4+4$

![Image of a tree with weights and partial sums]
The Iteration and its Computational Cost

• Current reservoir of unbiased estimates, threshold $z$
  – Reservoir maintained in two BSTs;
    • One for small items ($x_i \leq z$) and large ($x_i > z$)
    • Maintains partial sums for number and total weight of items $\leq x_i$

• Insert new weight
  – Binary tree search to find $t^* = \max\{ t: \sum_{i\leq t} x(i) / x(t) \geq t-1\}$
    • Find $\sum_{i\leq t} x(i)$ and $t$ by adding up appropriate partial sums for weights and counts
  – Compute new threshold $z$ from $\sum_{i\leq t^*} x(i) / z = t^*-1$
  – Transfer items between small and large BSTs as needed

• Discard one item at random using discard probabilities
  – Generate random $r$ uniformly on $[0,1]$
  – Find smallest $d \leq t^*$ such at $\sum_{i\leq d} (1 - x(i) / z) \geq r$
    • Binary search on small items: sum is function of counts and weight sums

• Basic computational takeaway
  – Computing in BST gives $O(\log m)$ complexity per arriving item
  – Actually $O(\log \log m)$ averaged over $m$ arrivals
Summary: IPPS Sampling on Data Streams

• Motivation
  – Computer networking: storing sampled flow records for later analysis

• Ingredients:
  – Probability/Statistics, Algorithms, Data structures

• Statistical properties
  – Statistically optimal trade-off between sampling size and variance

• IPPS stream reservoir sampling
  – Iterative algorithm that maintains only current unbiased estimates in fixed size reservoir of size m

• Implementation
  – Used binary search trees to get $O(\log m)$ computational cost per arrival
Additional references

• Binary trees in general:
  – Goodrich: Data Structures and Algorithms in Python. Ch. 8
    • TAMU Library, online
  – These notes describe insertion and search; BST also provide for
deletion, rebalance and other operations.

• IPPPS Stream Sampling:
  – Cohen et. Al., Efficient Stream Sampling for Variance-Optimal