Subsampling

Uniformly sample the coordinates as nested subsets

\[ n = S_0 \supseteq S_1 \supseteq S_2 \supseteq \cdots \supseteq S_{\log_2 n} \]

Include each item from \( S_{i-1} \) in \( S_i \) independently with probability 1/2

\( x_{S_i} \) is \( x \) restricted to coordinates in \( S_i \)

Subsampling in a Stream

Stream update \( e_i \leftarrow e_i + \Delta_i \)

If \( i \in S_0, i \in S_1 \) but \( i \notin S_2 \), feed \( i \) to first 2 algorithms but not to any other algorithms

Algorithm for Finding a Non-Zero Item

- If \( x \) has \( k \) non-zero entries, what’s the expected number of non-zero entries in \( x_{S_i} \)?
  - For each non-zero entry \( j \) in \( x \), let \( Z_j = 1 \) if \( j \in S_i \), and \( Z_j = 0 \) otherwise
  - \( E[Z] = \sum Z = k \cdot \frac{1}{2^i} \)
  - \( \text{Var}[Z] = \sum \text{Var}[Z] = k \cdot \text{Var}[Z_1] = k \cdot \left( \frac{1}{2^i} \right) \left( 1 - \frac{1}{2^i} \right) \leq \frac{k}{2^i} \)
  - If \( i = \lfloor \log_2 k \rfloor - 5 \), then \( 32 \leq E[Z] < 64 \) and \( \text{Var}[Z] < 64 \)
  - By Chebyshev’s inequality, \( \Pr[|Z - E[Z]| \geq 32] \leq \frac{\text{Var}[Z]}{32^2} \leq \frac{1}{16} \)
  - If we run a \( k' \)-sparse algorithm with \( k' = 96 \) on \( x_{S_i} \), we recover a non-zero item of \( x_{S_i} \) with probability at least \( 1-1/16 - 1/10 > 4/5 \), or output \( \text{FAIL} \)
  - But we don’t know \( i \)?

Algorithm for Finding a Non-Zero Item

- Run a \( k' = 96 \)-sparse vector algorithm on every \( x_{S_i} \)!
  - For each \( x_{S_i} \), our algorithm either returns a non-zero item of \( x_{S_i} \), and hence of \( x \), or outputs \( \text{FAIL} \)
  - For \( i = \lfloor \log_2 k \rfloor - 5 \), with probability at least \( 4/5 \), we output a non-zero item of \( x_{S_i} \), and hence of \( x \)
  - Space is \( (\log_2 n) \cdot O(k' \log n) = O(\log^2 n) \) bits!
    - (need to store \( S_0, \ldots, S_{\log_2 n} \) but can use hash function for these)
Outline

• Sketching Model
  • Estimating the Euclidean norm of a vector
  • Finding a non-zero coordinate of a vector

• Graph sketching
  • Boruvka’s spanning tree algorithm
  • Finding a spanning tree from a sketch

Sketching Graphs

Are there sketches for graphs? $A_G$ is the $n \times n$ adjacency matrix of a graph $G$

$$(\begin{pmatrix}A_G & S \end{pmatrix} \begin{pmatrix}A_G \end{pmatrix}) = \begin{pmatrix}SA_G \end{pmatrix}$$

• Is there a distribution on matrices $S$ with a small number of rows so that you can output a spanning tree of $G$, given $SA_G$, with high probability?

Application: Graph Streams

• Process a graph stream and see the edges of a graph $e_1, \ldots, e_m$ in an arbitrary order

  $e_3 e_10 e_1 e_2 e_5 e_7 e_6 \ldots$

• Make 1 pass over the stream
• Could store stream using $O(n^2)$ bits of memory
• Can we use only $n \cdot \text{poly}(\log n)$ bits of memory?
• How would you compute a spanning forest?

Computing a Spanning Forest

• For each edge $e$ in the stream
  • If $\text{________________________}$, store edge $e$
  • $\text{________________________}$ is “doesn’t form a cycle”
• Store at most $n-1$ edges, so $O(n \log n)$ bits of memory
• But what if you are allowed to delete edges? This is called a dynamic stream
Handling Deletions with Sketching

- Given $S \cdot A_G$, if $e$ is deleted, replace it with $S \cdot A_G - S \cdot A_e = S \cdot A_{G-e}$

- Memory to store $S \cdot A_G$ is $(\# \text{ of rows of } S) \cdot n \cdot \log n$ bits
  - Also need to store $S$, which is $(\# \text{ of rows of } S) \cdot n \cdot \log n$ bits

- **Goal:** find $S$ with a small # of rows so that given $S \cdot A_G$, can output a spanning tree of $G$ with high probability

- **Theorem:** there is a distribution on $S$ with $O(\log^2 n)$ rows!

---

Outline

- Sketching Model
  - Estimating the Euclidean norm of a vector
  - Finding a non-zero coordinate of a vector

- Graph sketching
  - Boruvka’s spanning tree algorithm
  - Finding a spanning tree from a sketch

---

Parallel Computing

- **Boruvka’s Spanning Tree Algorithm (Modified)**
  - Input graph is unweighted and connected
  - Initialize edgeset $E'$ to $\emptyset$
  - Create a list of $n$ groups of vertices, each initialized to a single vertex
  - While the list has more than one group
    - For each group $G$, include in $E'$ an edge $e$ from a vertex in $G$ to a vertex not in $G$
    - Merge groups connected by an edge in the previous step
  - Find a spanning tree among the edges in $E'$
Round 1

Group J

Round 1

Edge J-H

Round 1 Ends

List of Edges Added

• A-D
• B-A
• C-F
• D-B
• E-C
• F-C
• G-E
• H-J

• I-G
• J-H

Groups at Beginning of Round 2

List of Groups

• D-A-B
• F-C-E-G-I
• H-J
Round 2 Group H-J

Round 2 Edge J-I

List of Edges Added

- B-C
- I-J
- J-I

Spanning Tree Input Graph
Analysis

- If there are at least 2 groups in an iteration, then each group has an outgoing edge
  - Else, graph is disconnected

- If t groups at start of an iteration, at most t/2 groups at end of iteration
  - Consider graph with vertices $G_1, G_2, ..., G_t$ and r edges, where edges correspond to the groups we connect
  - Number of groups now at most number of connected components in $H$. Why?

- After $\log_2 n$ iterations, one group left
  - At most $n + n/2 + n/4 + ... + 1 \leq 2n$ edges in $E'$

- $E'$ contains a spanning tree
  - Invariant: the vertices in a group are connected

Outline

- Sketching Model
  - Estimating the Euclidean norm of a vector
  - Finding a non-zero coordinate of a vector

- Graph sketching
  - Boruvka's spanning tree algorithm
  - Finding a spanning tree from a sketch

Representing a Graph

- For node i, let $a_i$ be a vector indexed by node pairs

- If {i,j} is an edge, $a_i[i,j] = 1$ if j > i, and $a_i[i,j] = -1$ if j < i

- If {i,j} is not an edge, $a_i[i,j] = 0$

- Lemma: for a subset $S$ of nodes, $\text{Support}(\sum_{i \in S} a_i) = E(S, V \setminus S)$

- Proof: for edge {i,j}, if i, j $\in S$, the sum of entries on {i,j}-th column is 0
Spanning Tree Algorithm

• If we delete edge \((1, 2)\) in the stream, then \(a_1\) and \(a_2\) become:
  • \(a_1 = (0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0)\)
  • \(a_2 = (0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0)\)

• If we insert or delete edge \((i,j)\) we just update \(a_i\) and \(a_j\) accordingly

• But we can’t write down \(a_i\) since it is \(\Theta(n^2)\)-dimensional

• Million dollar question: what can we do to a vector if we can’t store it?

Spanning Tree Algorithm

• Store a sketch of each \(a_j\)

• We’ll need more than 1 sketch of each \(a_j\), let’s take \(O(\log n)\) sketches

• Maintain \(O(\log n)\) sketches \(C_1 \cdot a_j, \ldots, C_O(\log n) \cdot a_j\) for each \(a_j\) in a stream

• \(C_i\) squashes \(a_i\) down to \(O(\log^2 n)\) bits

• \(C_i \cdot a_j\) returns a non-zero item of \(a_j\) with probability \(4/5\), or returns FAIL

• A non-zero item of \(a_j\) is just an edge incident to vertex \(j\)

Spanning Tree Algorithm

• Compute \(O(\log n)\) sketches \(C_1 \cdot a_j, \ldots, C_O(\log n) \cdot a_j\) for each \(a_j\)

• \(C_i \cdot a_j\) outputs a non-zero item of \(a_j\) with probability \(> 4/5\), or returns FAIL

• Idea: Run Boruvka’s algorithm on sketches!

• For each node \(j\), use \(C_i \cdot a_j\) to get incident edge on \(j\)

• For \(i = 2, \ldots, O(\log n)\)
  • To get incident edge on group \(G \subseteq V\), use

  \[
  \sum_{j \in S} C_i a_j = C_i \left( \sum_{j \in S} a_j \right) \rightarrow e \subseteq \text{support}(\sum_{j \in S} a_j) = E(S, V \setminus S)
  \]

Spanning Tree Wrapup

• \(O(n \log n)\) sketches \(C_i \cdot a_j\) as \(i\) and \(j\) vary, so \(O(n \log^2 n)\) bits of space

• Note: a \(1/5\) fraction of sketches fail in each iteration in expectation, but a \(4/5\) fraction of groups get connected with other groups

• Expected number of iterations still \(O(\log n)\)

• Since sketches are linear, can maintain with insertions and deletions of edges

• Overall, \(O(n \log^2 n)\) bits of space to output a spanning tree!