Big-Data Algorithms: Overview

Reference: http://www.sketchingbigdata.org/fall17/lec/lec1.pdf

What's the problem here?

- So far, linear (i.e., linear-cost) algorithms have been "gold standard".
- What if linear algorithms aren't good enough?

What's the problem here?

- So far, linear (i.e., linear-cost) algorithms have been "gold standard".
- What if linear algorithms aren't good enough?
  Example: Search the web for pages of interest.

Topics of Interest

- **Sketching:** Compression of a data set that allows queries.
  - Compression $C(x)$ of some data set x that allows us to query $f(x)$.
  - May want to compute $f(x, y)$ from $C(x)$ and $C(y)$.
  - May want *composable* compression: if $x = x_1 x_2 \dots x_n$, would like to compute $C(x_1 x_2 \dots x_n x_{n+1}) = C(x\, x_{n+1})$ using just $C(x)$ and x_{n+1}.

Topics of Interest

- **Sketching:** Compression of a data set that allows queries.
  - Compression $C(x)$ of some data set x that allows us to query $f(x)$.
  - May want to compute $f(x, y)$ from $C(x)$ and $C(y)$.
  - May want *composable* compression: if $x = x_1 x_2 \ldots x_n$, would like to compute $C(x_1 x_2 \ldots x_n x_{n+1}) = C(x\, x_{n+1})$ using just $C(x)$ and x_{n+1}.
- **Streaming:** May not be able to store a huge dataset. Need to process *stream* of data, coming in one chunk at a time, on the fly. Must answer queries with sublinear memory.

Topics of Interest

- **Sketching:** Compression of a data set that allows queries.
  - Compression $C(x)$ of some data set x that allows us to query $f(x)$.
  - May want to compute $f(x, y)$ from $C(x)$ and $C(y)$.
  - May want *composable* compression: if $x = x_1 x_2 \ldots x_n$, would like to compute $C(x_1 x_2 \ldots x_n x_{n+1}) = C(x\, x_{n+1})$ using just $C(x)$ and x_{n+1}.
- **Streaming:** May not be able to store a huge dataset. Need to process *stream* of data, coming in one chunk at a time, on the fly. Must answer queries with sublinear memory.
- **Dimensionality reduction:** For example, spam filtering. Bag-of-words model: Let d be a dictionary of words. Represent email by vector v, where v_i is the number of times d_i appears in msg. Then $\dim v = |d|$.

- **Large-scale matrix computation**, such as *least squares regression*: Suppose we want to learn $f : \mathbb{R}^n \to \mathbb{R}$, where $f = \langle \mathbf{b}, \cdot \rangle$ for some $\mathbf{b} \in \mathbb{R}^n$, where

$$\langle \mathbf{u}, \mathbf{v} \rangle = \sum_{j=1}^{n} u_i v_i \qquad \forall\, \mathbf{u}, \mathbf{v} \in \mathbb{R}^n.$$

  Collect data $\{\, (\mathbf{x}_i \in \mathbb{R}^n, y_i \in \mathbb{R}) : 1 \leq i \leq m \,\}$.
  Want to compute $\mathbf{b}$ minimizing

$$\|\mathbf{X}\mathbf{b} - \mathbf{y}\|_2^2 = \left(\sum_{j=1}^{n} (y_i - \langle \mathbf{b}, \mathbf{x}_i \rangle)^2 \right)^{1/2},$$

  where $\mathbf{X} \in \mathbb{R}^{m \times n}$ is composed of the (column) vectors $\mathbf{x}_1^T, \ldots, \mathbf{x}_m^T$ and $\| \cdot \|_2 = \sqrt{\langle \cdot, \cdot \rangle}$ is ℓ_2-norm.

  Also, principal component analysis, given by singular value decomposition of matrix: which features are most important?

Approximate Counting

Problem: Monitor a sequence of events, allow approximate count of number of events so far at any time.

Approximate Counting

Problem: Monitor a sequence of events, allow approximate count of number of events so far at any time.
Create data structure maintaining a single integer n (initialize to zero) and supporting the operations

- `init()`: set $n \leftarrow 0$.
- `update()`: increments n.
- `query()`: prints (estimate of) n

Approximate Counting

Problem: Monitor a sequence of events, allow approximate count of number of events so far at any time.

Create data structure maintaining a single integer n (initialize to zero) and supporting the operations

- `init()`: set $n \leftarrow 0$.
- `update()`: increments n.
- `query()`: prints (estimate of) n

Why approximation?

If we want exact value, then can store n via a counter, a sequence of $\lceil \log n \rceil$ bits ("log" is "$\log_2$").

Approximate Counting

Problem: Monitor a sequence of events, allow approximate count of number of events so far at any time.
Create data structure maintaining a single integer n (initialize to zero) and supporting the operations

- `init()`: set $n \leftarrow 0$.
- `update()`: increments n.
- `query()`: prints (estimate of) n

Why approximation?
If we want exact value, then can store n via a counter, a sequence of $\lceil \log n \rceil$ bits ("log" is "$\log_2$").
Can't do better:
If we use $f(n)$ bits to store n, then there are $2^{f(n)}$ configurations.
To store exact value of all integers up to n, must have

$$2^{f(n)} \geq n$$

Approximate Counting

Problem: Monitor a sequence of events, allow approximate count of number of events so far at any time.
Create data structure maintaining a single integer n (initialize to zero) and supporting the operations

- `init()`: set $n \leftarrow 0$.
- `update()`: increments n.
- `query()`: prints (estimate of) n

Why approximation?
If we want exact value, then can store n via a counter, a sequence of $\lceil \log n \rceil$ bits ("log" is "$\log_2$").
Can't do better:
If we use $f(n)$ bits to store n, then there are $2^{f(n)}$ configurations.
To store exact value of all integers up to n, must have

$$2^{f(n)} \geq n \implies f(n) \geq \log n$$

Approximate Counting

Problem: Monitor a sequence of events, allow approximate count of number of events so far at any time.
Create data structure maintaining a single integer n (initialize to zero) and supporting the operations

- `init()`: set $n \leftarrow 0$.
- `update()`: increments n.
- `query()`: prints (estimate of) n

Why approximation?
If we want exact value, then can store n via a counter, a sequence of $\lceil \log n \rceil$ bits ("log" is "$\log_2$").
Can't do better:
If we use $f(n)$ bits to store n, then there are $2^{f(n)}$ configurations.
To store exact value of all integers up to n, must have

$$2^{f(n)} \geq n \implies f(n) \geq \log n \implies f(n) \geq \lceil \log n \rceil$$

Approximate Counting

Problem: Monitor a sequence of events, allow approximate count of number of events so far at any time.
Create data structure maintaining a single integer n (initialize to zero) and supporting the operations

- `init()`: set $n \leftarrow 0$.
- `update()`: increments n.
- `query()`: prints (estimate of) n

Why approximation?
If we want exact value, then can store n via a counter, a sequence of $\lceil \log n \rceil$ bits ("log" is "$\log_2$").
Can't do better:
If we use $f(n)$ bits to store n, then there are $2^{f(n)}$ configurations.
To store exact value of all integers up to n, must have

$$2^{f(n)} \geq n \implies f(n) \geq \log n \implies f(n) \geq \lceil \log n \rceil \qquad \text{since } n \in \mathbb{Z}$$

If we want sublinear-space algorithm, need an estimate $\tilde{n}$ of n.
Want to know that for some $\varepsilon, \delta \in (0, 1)$, we have

$$\mathbb{P}\left(|\tilde{n} - n| > \varepsilon\, n\right) < \delta.$$

If we want sublinear-space algorithm, need an estimate $\tilde{n}$ of n.
Want to know that for some $\varepsilon, \delta \in (0,1)$, we have

$$\mathbb{P}\left(|\tilde{n} - n| > \varepsilon\, n\right) < \delta.$$

Equivalently:

$$\mathbb{P}\left(|\tilde{n} - n| \leq \varepsilon\, n\right) \geq 1 - \delta.$$

Morris' algorithm: Uses an integer counter X, with data structure operations

- `init()`: sets $X \leftarrow 0$
- `update()`: increments X with probability 2^{-X}
- `query()`: outputs $\tilde{n} = 2^X - 1$

Intuitively, X attempts to store a value approximately $\log n$.

How good is this?

Morris' algorithm: Uses an integer counter X, with data structure operations

- `init()`: sets $X \leftarrow 0$
- `update()`: increments X with probability 2^{-X}
- `query()`: outputs $\tilde{n} = 2^X - 1$

Intuitively, X attempts to store a value approximately $\log n$.

How good is this? Not so great; we'll see that

$$\mathbb{P}\left(|\tilde{n} - n| > \varepsilon\, n\right) < \frac{1}{2\varepsilon^2}$$

Since $\varepsilon < 1$, RHS exceeds $\frac{1}{2}$, which means that estimator may always be zero!

Improvement Morris+: Create s independent copies of Morris, and average their outputs. Calling these estimators $\tilde{n}_1, \ldots, \tilde{n}_s$, then output is

$$\tilde{n} = \frac{1}{s} \sum_{i=1}^{n} \tilde{n}_i.$$

Then

$$\mathbb{P}\left(|\tilde{n} - n| > \varepsilon\, n\right) < \frac{1}{2s\varepsilon^2}$$

So

$$\mathbb{P}\left(|\tilde{n} - n| > \varepsilon\, n\right) < \delta \qquad \text{for } s > \frac{1}{2\varepsilon^2\delta} = \Theta(1/\delta)$$

Better!

Improvement Morris++: Reduces dependence of failure probability from $\Theta(1/\delta)$ to $\Theta(\log 1/\delta)$.

Improvement Morris++: Reduces dependence of failure probability from $\Theta(1/\delta)$ to $\Theta(\log 1/\delta)$.

Run t instances of Morris+, each with failure probability $\frac{1}{3}$. So $s = \Theta(1/\varepsilon^2)$ for each instance. Now output median estimate of these t Morris+ instances. Calling this output $\tilde{n}$, it turns out that

$$\mathbb{P}\left(|\tilde{n} - n| > \varepsilon\, n\right) < \delta \qquad \text{for } t = \Theta(\log 1/\delta).$$

Probability Review

Let X be a random variable taking values in $S \subseteq \mathbb{R}$.
The *expected value* of X is

$$\mathbb{E}X = \sum_{j \in S} j \cdot \mathbb{P}(X = j).$$

The *variance of* X is

$$\mathrm{Var}[X] = \mathbb{E}\big((X - \mathbb{E}X)^2\big).$$

Linearity of expected value: Let X and Y be random variables. Than

$$\mathbb{E}(aX + bY) = a\,\mathbb{E}X + b\,\mathbb{E}Y \qquad \forall\, a, b \in \mathbb{R}.$$

Markov's inequality: If X is a nonnegative random variable, then

$$\mathbb{P}(X > \lambda) < \frac{\mathbb{E}X}{\lambda} \qquad \forall\, \lambda > 0.$$

Chebyshev's inequality: Let X be a nonnegative random variable. Then

$$\mathbb{P}(|X - \mathbb{E}X| > \lambda) < \frac{\mathbb{E}(X - \mathbb{E}X)^2}{\lambda^2} = \frac{\mathrm{Var}[X]}{\lambda^2} \qquad \forall\, \lambda > 0.$$

More generally, if $p \geq 1$, then

$$\mathbb{P}(|X - \mathbb{E}X| > \lambda) < \frac{\mathbb{E}(X - \mathbb{E}X)^p}{\lambda^p}. \qquad \forall\, \lambda > 0.$$

Chernoff's inequality: Suppose $X_1, X_2, \ldots, X_n$ are independent random variables with $X_i \in [0, 1]$. Let $X = \sum_{i=1}^n X_i$. Then

$$\mathbb{P}(|X - \mathbb{E}X| > \varepsilon\, \mathbb{E}X) \leq 2 \cdot \mathrm{e}^{-\varepsilon^2 \mu/3} \qquad \forall\, \varepsilon \in (0, 1).$$

Analysis of Morris' algorithm

Let X_n be X after n updates.

Claim: $\mathbb{E}2^{X_n} = n + 1$ for $n \in \mathbb{N}_0$.

Proof of claim: By induction, the base case $n = 0$ being

$$\mathbb{E}2^{X_n} = \mathbb{E}2^{X_0} = \mathbb{E}1 = n + 1.$$

Induction step: Suppose that $\mathbb{E}2^{X_n} = n+1$ for some $n \in \mathbb{N}_0$. Then

$$\begin{aligned}
\mathbb{E}2^{X_{n+1}} &= \sum_{j=0}^{\infty} \mathbb{P}(X_n = j) \cdot \mathbb{E}(2^{X_{n+1}} \mid X_n = j) \\
&= \sum_{j=0}^{\infty} \mathbb{P}(X_n = j) \cdot \left(\left(1 - \frac{1}{2^j}\right) 2^j + \frac{1}{2^j} \cdot 2^{j+1} \right) \\
&= \sum_{j=0}^{\infty} \mathbb{P}(X_n = j)\, 2^j + \sum_{j=0}^{\infty} \mathbb{P}(X_n = j) \\
&= \mathbb{E}2^{X_n} + 1 \\
&= (n+1) + 1,
\end{aligned}$$

as required.

So $\tilde{n} = 2^X - 1$ is an unbiased estimator of n.
Need to find its variance. Using Chebyshev:

$$\mathbb{P}(|\tilde{n} - n| > \varepsilon n) < \frac{1}{\varepsilon^2 n^2} \cdot \mathbb{E}(\tilde{n} - n)^2 = \frac{1}{\varepsilon^2 n^2} \cdot \mathbb{E}(2^X - 1 - n)^2.$$

Claim: $\mathbb{E}2^{2X_n} = \frac{3}{2}n^2 + \frac{3}{2}n + 1$ for $n \in \mathbb{N}_0$.

Proof: By induction, the base case $n = 0$ being

$$\mathbb{E}2^{2X_0} = \mathbb{E}2^0 = 1 = \tfrac{3}{2} \cdot 0^2 + \tfrac{3}{2} \cdot 0 + 1.$$

For the inductive step, suppose that $\mathbb{E}2^{2X_n} = \frac{3}{2}n^2 + \frac{3}{2}n + 1$ for some $n \in \mathbb{N}_0$. Then

$$\begin{aligned}
\mathbb{E}2^{2X_{n+1}} &= \sum_{j=0}^{\infty} \mathbb{P}(2^{X_n} = j) \cdot \mathbb{E}(2^{2X_{n+1}} \mid 2^{X_n} = j) \\
&= \sum_{j=0}^{\infty} \mathbb{P}(2^{X_n} = j) \cdot \left(\frac{1}{j} \cdot 4j^2 + \left(1 - \frac{1}{j}\right) \cdot j^2 \right) \\
&= \sum_{j=0}^{\infty} \mathbb{P}(2^{X_n} = j) \cdot (j^2 + 3j) \\
&= \mathbb{E}2^{2X_n} + 3 \cdot \mathbb{E}2^{X_n} \\
&= \left(\tfrac{3}{2}n^2 + \tfrac{3}{2}n + 1\right) + 3(n+1) \\
&= \tfrac{3}{2}(n+1)^2 + \tfrac{3}{2}(n+1) + 1,
\end{aligned}$$

as required.

Since $\mathrm{Var}[Z] = \mathbb{E}[Z^2] - (\mathbb{E}[Z])^2$ for any random variable Z, we have

$$\mathbb{P}(|\tilde{n} - n| > \varepsilon n) < \frac{1}{\varepsilon^2 n^2} \cdot \frac{n^2}{2} = \frac{1}{2\varepsilon^2},$$

as claimed for (the original version of) Morris.

Morris+: As on earlier slide.

Morris++: Run t instances of Morris+, each with failure probability $\frac{1}{3}$. So $s = \Theta(1/\varepsilon^2)$ for each instance. Now output median estimate of these t Morris+ instances.

Morris+: As on earlier slide.

Morris++: Run t instances of Morris+, each with failure probability $\frac{1}{3}$. So $s = \Theta(1/\varepsilon^2)$ for each instance. Now output median estimate of these t Morris+ instances.

Expected number of unsuccessful Morris+ instantiations: $\frac{1}{3}t$.
Expected number of successful Morris+ instantiations: $\frac{2}{3}t$.

Morris+: As on earlier slide.

Morris++: Run t instances of Morris+, each with failure probability $\frac{1}{3}$. So $s = \Theta(1/\varepsilon^2)$ for each instance. Now output median estimate of these t Morris+ instances.

Expected number of unsuccessful Morris+ instantiations: $\frac{1}{3}t$.
Expected number of successful Morris+ instantiations: $\frac{2}{3}t$.

If median is bad estimate, then at most half of the Morris+ instantiations can succeed.
Hence number of succeeding instantiations deviated from its expectation by at least $\frac{1}{2} \cdot \frac{1}{3}t = \frac{1}{6}t$.

For $i \in \{1, \ldots, t\}$, define the random variable

$$Y_i = \begin{cases} 1 & \text{if } i\text{th Morris+ instantiation succeeds,} \\ 0 & \text{if } i\text{th Morris+ instantiation fails.} \end{cases}$$

For $i \in \{1, \ldots, t\}$, define the random variable

$$Y_i = \begin{cases} 1 & \text{if } i\text{th Morris+ instantiation succeeds,} \\ 0 & \text{if } i\text{th Morris+ instantiation fails.} \end{cases}$$

So

$$\mathbb{P}\left(\sum_{i=1}^{t} Y_i \leq \frac{t}{2}\right) \leq$$

For $i \in \{1, \ldots, t\}$, define the random variable

$$Y_i = \begin{cases} 1 & \text{if } i\text{th Morris+ instantiation succeeds,} \\ 0 & \text{if } i\text{th Morris+ instantiation fails.} \end{cases}$$

So

$$\mathbb{P}\left(\sum_{i=1}^{t} Y_i \leq \frac{t}{2}\right) \leq \mathbb{P}\left(\left|\sum_{i=1}^{t} Y_i - \mathbb{E}\sum_{i=1}^{t} Y_i\right| \geq \frac{t}{6}\right)$$

For $i \in \{1, \dots, t\}$, define the random variable

$$Y_i = \begin{cases} 1 & \text{if } i\text{th Morris+ instantiation succeeds,} \\ 0 & \text{if } i\text{th Morris+ instantiation fails.} \end{cases}$$

So

$$\mathbb{P}\left(\sum_{i=1}^{t} Y_i \leq \frac{t}{2}\right) \leq \mathbb{P}\left(\left|\sum_{i=1}^{t} Y_i - \mathbb{E}\sum_{i=1}^{t} Y_i\right| \geq \frac{t}{6}\right) \leq 2e^{-t/3},$$

the last by Chernoff's inequality.

For $i \in \{1, \ldots, t\}$, define the random variable

$$Y_i = \begin{cases} 1 & \text{if } i\text{th Morris+ instantiation succeeds,} \\ 0 & \text{if } i\text{th Morris+ instantiation fails.} \end{cases}$$

So

$$\mathbb{P}\left(\sum_{i=1}^{t} Y_i \leq \frac{t}{2}\right) \leq \mathbb{P}\left(\left|\sum_{i=1}^{t} Y_i - \mathbb{E}\sum_{i=1}^{t} Y_i\right| \geq \frac{t}{6}\right) \leq 2e^{-t/3},$$

the last by Chernoff's inequality. Now

$$2e^{t/3} < \delta \iff t > 3\log\frac{1}{2\delta} = \Theta\left(\log\frac{1}{\delta}\right).$$

For $i \in \{1, \ldots, t\}$, define the random variable

$$Y_i = \begin{cases} 1 & \text{if } i\text{th Morris+ instantiation succeeds,} \\ 0 & \text{if } i\text{th Morris+ instantiation fails.} \end{cases}$$

So

$$\mathbb{P}\left(\sum_{i=1}^{t} Y_i \le \frac{t}{2}\right) \le \mathbb{P}\left(\left|\sum_{i=1}^{t} Y_i - \mathbb{E}\sum_{i=1}^{t} Y_i\right| \ge \frac{t}{6}\right) \le 2e^{-t/3},$$

the last by Chernoff's inequality. Now

$$2e^{t/3} < \delta \iff t > 3\log\frac{1}{2\delta} = \Theta\left(\log\frac{1}{\delta}\right).$$

So

$$\mathbb{P}\left(\sum_{i=1}^{t} Y_i \le \frac{t}{2}\right) < \delta \qquad \text{for } t = \Theta\left(\log\frac{1}{\delta}\right).$$

as required.