

An Optimal Algorithm for the Distinct Elements Problem

Daniel M. Kane

Department of Mathematics
Stanford University
aladkeenin@gmail.com

February 21st, 2014

Joint work with Jelani Nelson (Harvard) and David Woodruff (IBM)

Streaming Algorithms

Algorithm to analyze

- Data passing through a server OR
- Data in a large database

In either case, there may be too much data to store a full (separate) copy of everything. Want an algorithm that:

- Uses little space
- Little processing time per item
- Makes only one pass through the data

Distinct Elements Problem

Distinct elements problem: Given a stream S of elements of $[n] := \{1, 2, \dots, n\}$, return the number of distinct elements in S .

Example:

$$S = 1, 7, 2, 7, 23, 1, 7$$

Distinct elements are 1, 2, 7, 23. Return 4.

Applications:

- Detecting Denial of Service Attacks
[Akella, Bharambe, Reiter, Seshan]
- Query planning
[Selinger, Astrahan, Chamberlin, Lorie]
- Data Integration
[Brown, Haas, Myllymaki, Pirahesh, Reinwald, Sismanis]

Approximation

Returning $|S|$ exactly is too much to hope for.

- Compare $|S|, |S \cup \{x\}|$ to determine if $x \in S$
- Would essentially need to store all elements of the stream
- Requires $\Omega(n)$ space

Instead return answer F so that $F = |S|(1 \pm \epsilon)$ for some small $\epsilon > 0$.

Randomization

Deterministic algorithms also do not work:

- Let S_i be subsets that overlap at most $2/3$ of their elements.
- Need to distinguish stream $S_i S_i$ from $S_i S_j$.
- Need to store enough information to tell which S_i you have.
- Requires $\Omega(n)$ space.

Thus, to obtain reasonable space, we will need to use a randomized algorithm that we will only require to be correct with probability $2/3$ (this can be improved to $1 - \delta$ by running $O(\log(\delta^{-1}))$ copies of the algorithm in parallel).

Problem Statement

Problem

Find a randomized algorithm that given a stream S of elements in $[n]$:

- *Makes only a single pass over the stream*
- *Uses s space*
- *Uses t time per update*

Returns a value in $(|S|(1 - \epsilon), |S|(1 + \epsilon))$ with probability at least $2/3$ rd

Past Work

Paper	Space	Time	Notes
[FM85]	$O(\log(n))$	-	Const. ϵ , rand. oracle
[AMS99]	$O(\log(n))$	$O(\log(n))$	Const. ϵ
[GT01]	$O(\epsilon^{-2} \log(n))$	$O(\epsilon^{-2})$	
[BJKST02]	$O(\epsilon^{-2} \log(n))$	$O(\log(\epsilon^{-1}))$	
[BJKST02]	$\tilde{O}(\epsilon^{-2} + \log(n))$	$\tilde{O}(\epsilon^{-2} \log \log(n))$	
[KNW10]	$O(\epsilon^{-2} + \log(n))$	$O(1)$	Optimal, This talk

Lower Bounds

Space lower bounds of $\Omega(\epsilon^{-2} + \log(n))$ were proven by Indyk and Woodruff (later extended to cover all $\epsilon > n^{-1/2}$ by Woodruff).

Idea: Given a stream ST need to distinguish the cases:

- S and T overlap on a $1/2 + \epsilon$ -fraction of their entries
- S and T overlap on a $1/2 - \epsilon$ -fraction of their entries

This reduces it to the one-way communication complexity of the Gap-Hamming Problem.

Basic Idea: Subsampling

The basic idea behind the majority of algorithms is due to Flajolet and Martin.

- Let A be a random subset of $[n]$ of size m
- Consider $|S \cap A|$
 - ▶ Has expectation $\frac{|S|m}{n}$
 - ▶ Is 0 with $5/6$ probability if $|S| < \frac{n}{6m}$
 - ▶ Is positive with $5/6$ probability if $|S| > \frac{2n}{m}$

Idea: Keep track of whether or not S contains any element of A for a number of random A of different sizes.

Implementation

Let $h : [n] \rightarrow [n]$ be a random hash function.

Let $\text{lsb}(x)$ be the location of the least significant bit of x which is a 1.

- Note that $A_k := \{x \in [n] : \text{lsb}(h(x)) > k\}$ is a random subset of $[n]$ of size about $n2^{-k}$

Algorithm:

- Maintain a counter $r = \max_{x \in S}(\text{lsb}(h(x)))$
- With $2/3$ probability $\log(|S|) = r \pm 2$
- Thus 2^r gives an approximation of $|S|$ likely good to within a factor of 4
- Requires $O(\log \log(n))$ bits of space

Reducing Error: Statistics

To get a better approximation to $|S|$, you can run multiple copies of this in parallel:

- Take $K \approx \epsilon^{-2}$
- Have K hash functions h_i and record $r_i = \max_{x \in S} (\text{lsb}(h_i(x)))$
- Pick some $r_* \approx \log(|S|)$ (perhaps given by the value of an extra r_i)
- $\Pr(r_i \leq r_*) \approx \exp(-|S|2^{-r_*})$
- Let $T = \#\{i : r_i \leq r_*\}$
- Estimator:

$$|S| \approx 2^{r_*} \log \left(\frac{K}{T} \right)$$

Improving Runtime: One Bin per Element

In the previous algorithm, we needed to update the values of $K \approx \epsilon^{-2}$ bins per update, causing the update time to be about ϵ^{-2} . We would like to improve this to constant time per update.

Idea: Each update effects only one bin

- $K \approx \epsilon^{-2}$, have K counters C_i
- Hash functions $h : [n] \rightarrow [n]$ and $g : [n] \rightarrow [K]$
- When you see x , replace $C_{g(x)}$ with the maximum of its current value and $\text{lsb}(h(x))$
- So C_i holds the value $\max_{x \in S: g(x)=i} \text{lsb}(h(x))$.

Analysis: Balls and Bins

- Pick r_* so that $2^{r_*} \approx |S|/K$
- How many C_i store a value of more than r_* ?
- Let $A := \{x \in S : \text{lsb}(h(x)) > r_*\}$
- $|A| = |S|2^{-r_*}(1 \pm \epsilon)$ with reasonable probability
- We need to count the number of distinct values that g takes on A

Question: $|A| \approx K$ balls are randomly thrown into K bins. What can we say about the distribution of T , the number of bins with at least one ball in them?

Balls and Bins

Lemma

$$\mathbb{E}[T] = K \left(1 - \left(1 - \frac{1}{K} \right)^{|A|} \right).$$

Furthermore, if $100 < |A| < K/20$,

$$\text{Var}(T) < \frac{4|A|^2}{K}.$$

Thus, if $|A| \approx K \approx \epsilon^{-2}$,

$$T = K \left(1 - \left(1 - \frac{1}{K} \right)^{|A|} \right) (1 \pm \epsilon)$$

Or

$$|A| = \frac{\log(1 - T/K)}{\log(1 - 1/K)} (1 \pm \epsilon)$$

Which gives us an estimator for $|S| = |A|2^{r^*} (1 \pm \epsilon)$.

Improving Space: Store Count Differences

The previous algorithm needs to maintain $K \approx \epsilon^{-2}$ counters of $\log \log(n)$ bits each, which is more space than we would like. On the other hand, most counters store values very close to r_* .

Idea: Instead store the differences between C_i and r_*

- With high probability the number of counters differing from r_* by more than b will be $O(K2^{-b})$
- Total space necessary is $O(K) = O(\epsilon^{-2})$
- Need a Variable Bitlength Array (VBA) with constant time updates (one has been developed by Blandford and Blelloch)

Problem: Need 2^{r_*} to be a good approximation to $|S|/K$ throughout entire computation, not just at the end.

RoughEstimator

New algorithm: RoughEstimator maintain an approximation to $|S|$ so that with 90% probability is correct to within a constant factor *for the entire stream*.

- Same basic idea as above
- $K \approx \log^{2/3}(n)$
- Hash functions $h : [n] \rightarrow [n]$, $g : [n] \rightarrow [K]$
- Maintain $C_i = \max_{x \in S: g(x)=i} (\text{lsb}(h(x)))$
 - ▶ Total space $O(\log^{2/3}(n) \log \log(n))$
- Let r_* be the median of the C_i
- $|S| = \Theta(2^{r_*} K)$ with probability $1 - O(K^{-2})$
- Sufficient to verify correctness when $|S|$ is a power 2
- Correct at all such times with probability $1 - O(\log(n)^{-1/3})$

Space Efficient Version

- Run RoughEstimator in parallel
 - ▶ Maintain r_* with $2^{r_*} \approx |S|/K$
- Hash functions $h : [n] \rightarrow [n]$, $g : [n] \rightarrow [K]$
- Maintain $C_i = \max_{x \in S: g(x)=i} (\text{lsb}(h(x))) - r_*$ in VBA
- Total space $O(K + \log(n)) = O(\epsilon^{-2} + \log(n))$
- Constant time updates unless r_* changes
- Deamortize decrements to the C_i when r_* increases

Derandomization

Thus, we have an algorithm which uses $O(\epsilon^{-2} + \log(n))$ space and $O(1)$ update time given access to random functions h and g . Unfortunately, we don't have access to these. We will need to store a description of any hash functions that we use, which will add to our space bound.

We need to derandomize this algorithm. Namely, we need to find less random families of functions (which can be described in a small number of bits) that are still good enough for our analysis to work.

k -Independence

We will be able to do our derandomization with the help of k -wise independent families of hash functions. Recall:

Definition

A family \mathcal{H} of hash functions $h : U \rightarrow V$ is k -wise independent if for any distinct $u_1, \dots, u_k \in U$ and any $v_1, \dots, v_k \in V$

$$\Pr_{h \in \mathcal{H}}(h(u_i) = v_i \text{ for all } i) = \frac{1}{|V|^k}.$$

Or in other words, h acts like a completely random function when restricted to any k elements of U .

There are known (approximately) k -wise independent families of hash functions which can be stored in $O(k \log(|U||V|))$ bits and evaluated in constant time.

Analysis

Recall that the analysis of both our main algorithm and RoughEstimator depended on understanding the statistic

$$T = \#\{i : \exists x \in S, \text{lsb}(h(x)) > r_*, g(x) = i\}.$$

Or in other words, letting

$$A = \{x \in S : \text{lsb}(h(x)) > r_*\},$$

$$T = |g(A)|.$$

Enough to show that $|A| = 2^{-r_*}|S|(1 \pm \epsilon)$ and that

$$T = K \left(1 - \left(1 - \frac{1}{K}\right)^{|A|}\right) (1 \pm \epsilon) \text{ with large probability.}$$

2-Wise Independent h

We claim that choosing h from a 2-wise independent family is sufficient to control the size of A .

- $\mathbb{E}[|A|] = |S|2^{-r^*}$
- $\text{Var}(|A|) = O(|S|2^{-r^*})$
- Both of these still hold under 2-independence
- The bounds on $|A|$ follow from Chebyshev's inequality

k -Wise Independent Balls and Bins

- Want to control $T = |g(A)|$ under k -wise independence of g
- Under full independence had $\mathbb{E}[T] = K(1 - (1 - 1/K)^{|A|})$ and $\text{Var}(T) = O(|A|^2/K)$, which was enough
- Need to show the above under k -wise independence

k -Wise Independent Balls and Bins

Need to show that the expected size of $g(A)$ is approximately preserved by k -wise independence.

Lemma

Let A be a finite set and K a positive integer. Let $g, g' : A \rightarrow [K]$ be functions with g random and g' chosen from a k -wise independent family. Let $T = |g(A)|$, $T' = |g'(A)|$, then

$$|\mathbb{E}[T] - \mathbb{E}[T']| \leq K \left(\frac{|A|}{K}\right)^k / k!$$

Thus k -wise independence will be sufficient to preserve our expectations for $k \approx \log(\epsilon^{-1})$.

Considering Each Bin

- Let T_i be the indicator random variable for the event that i is in the image of g , and T'_i the indicator of the event that i is in the image of g'
- $T = \sum_{i=1}^K T_i, T' = \sum_{i=1}^K T'_i$
- Enough to show that

$$|\mathbb{E}[T_i] - \mathbb{E}[T'_i]| \leq \left(\frac{|A|}{K}\right)^k / k!$$

Approximate Inclusion-Exclusion

For each $x \in A$, let B_x be the event that $g'(x) = i$.

Then T'_i is the indicator function of the event $B = \bigvee_{x \in A} B_x$.

Theorem (Inclusion-Exclusion)

$$\Pr(B) = \sum_{t=1}^{|A|} (-1)^{t+1} \sum_{\{x_1, \dots, x_t\} \subset A} \Pr(B_{x_1} \wedge B_{x_2} \wedge \dots \wedge B_{x_t}).$$

Approximate Inclusion-Exclusion

For each $x \in A$, let B_x be the event that $g'(x) = i$.

Then T'_i is the indicator function of the event $B = \bigvee_{x \in A} B_x$.

Theorem (Approximate Inclusion-Exclusion)

If m is odd,

$$\Pr(B) \leq \sum_{t=1}^m (-1)^{t+1} \sum_{\{x_1, \dots, x_t\} \subset A} \Pr(B_{x_1} \wedge B_{x_2} \wedge \dots \wedge B_{x_t}).$$

If m is even,

$$\Pr(B) \geq \sum_{t=1}^m (-1)^{t+1} \sum_{\{x_1, \dots, x_t\} \subset A} \Pr(B_{x_1} \wedge B_{x_2} \wedge \dots \wedge B_{x_t}).$$

Bounds on $\Pr(B)$

Thus, $\mathbb{E}[T'_i] = \Pr(B)$ is bounded between

$$\sum_{t=1}^{k-1} (-1)^{t+1} \sum_{\{x_1, \dots, x_t\} \subset A} \Pr(B_{x_1} \wedge B_{x_2} \wedge \dots \wedge B_{x_t}),$$

and

$$\sum_{t=1}^k (-1)^{t+1} \sum_{\{x_1, \dots, x_t\} \subset A} \Pr(B_{x_1} \wedge B_{x_2} \wedge \dots \wedge B_{x_t}).$$

Both of these bounds are determined by k -independence, and they differ by

$$\sum_{\{x_1, \dots, x_k\} \subset A} \Pr(B_{x_1} \wedge B_{x_2} \wedge \dots \wedge B_{x_k}) = \binom{|A|}{k} K^{-k} \leq \left(\frac{|A|}{K}\right)^k / k!$$

Since g is also k -wise independent, $\mathbb{E}[T_i]$ and $\mathbb{E}[T'_i]$ differ by at most this much.

Composition

- Unfortunately, there may not be enough space to store an $\log(\epsilon^{-1})$ -wise independent hash function from $[n]$ to $[K]$.
- To fix this let $g_1 : [n] \rightarrow [K^3]$ be 2-independent and $g_2 : [K^3] \rightarrow [K]$ be $\log(\epsilon^{-1})$ -wise independent and let $g = g_2 \circ g_1$.
 - ▶ With probability $1 - 1/K$, $|g_1(A)| = |A|$
 - ▶ If this holds, our analysis for $|g_2(g_1(A))|$ works

Final Algorithm: RoughEstimator

- **Initialize:**

- ▶ Let $K \approx \log^{2/3}(n)$, $k \approx \log(\log(n))$
- ▶ Let $h : [n] \rightarrow [n]$ be 2-independent, $g_1 : [n] \rightarrow [K^3]$ be 2-independent and $g_2 : [K^3] \rightarrow [K]$ k -independent, $g = g_2 \circ g_1$
- ▶ Initialize K counters of $\log \log(n)$ bits each C_i at 0

- **Update(x) :** $C_{g(x)} \leftarrow \max(C_{g(x)}, \text{lsb}(h(x)))$

- **Query:** Return $2^{\text{median}(C_i)} K$.

Final Algorithm

- **Initialize:**

- ▶ Let $K \approx \epsilon^{-2}$, $k \approx \log(\epsilon^{-1})$
- ▶ Let $h : [n] \rightarrow [n]$ be 2-independent, $g_1 : [n] \rightarrow [K^3]$ be 2-independent and $g_2 : [K^3] \rightarrow [K]$ k -independent, $g = g_2 \circ g_1$
- ▶ Initialize a copy of *RoughEstimator*
- ▶ Initialize a VBA storing C_1, \dots, C_K , all initially 0. Initialize $r = 0$.

- **Update(x) :**

- ▶ $C_{g(x)} \leftarrow \max(C_{g(x)}, \text{lsb}(h(x)) - r)$
- ▶ *Update(RoughEstimator)*
- ▶ Let $r' \leftarrow \lfloor \log(\text{Query}(\text{RoughEstimator})/K) \rfloor$
- ▶ If $r' \neq r$: $C_i \leftarrow C_i + r - r'$, $r \leftarrow r'$






- **Query:**







- ▶ Let $T = \#\{i : C_i > 0\}$
- ▶ Return

$$2^r \frac{\log(1 - T/K)}{\log(1 - 1/K)}$$

Conclusions

We have produced a streaming algorithm to compute approximations to the number of distinct elements in a data stream with asymptotically optimal space and update time. Similar techniques also yield a nearly optimal algorithm for L^0 moment estimation (where updates can remove copies elements from the stream as well as add them), although the correct space bound for that problem remains open.

-  A. Akella, A. Bharambe, M. Reiter, and S. Seshan *Detecting DDoS attacks on ISP networks* In Proc. MPDS, 2003.
-  N. Alon, Y. Matias, and M. Szegedy *The Space Complexity of Approximating the Frequency Moments* J. Comput. Syst. Sci., 58(1):137–147, 1999.
-  Z. Bar-Yossef, T.S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan *Counting distinct elements in a data stream* In Proc. RANDOM, pages 1–10, 2002.
-  D. K. Blandford and G. E. Blelloch *Compact dictionaries for variable-length keys and data with applications* ACM Trans. Alg., 4(2), 2008.
-  P. Brown, P. J. Haas, J. Myllymaki, H. Pirahesh, B. Reinwald, and Y. Sismanis *Toward automated large-scale information integration and discovery* In Data Management in a Connected World, pages 161–180, 2005.

-  P. Flajolet and G. N. Martin *Probabilistic counting algorithms for data base applications* J. Comput. Syst. Sci., 31(2):182–209, 1985.
-  P. B. Gibbons and S. Tirthapura *Estimating simple functions on the union of data streams* In Proc. SPAA, pages 281–291, 2001.
-  P. Indyk and D. P. Woodruff *Tight lower bounds for the distinct elements problem* In Proc. FOCS, pages 283–, 2003.
-  Daniel M. Kane, Jelani Nelson, David P. Woodruff *An Optimal Algorithm for the Distinct Elements Problem*, Symposium on Principles of Database Systems (PODS) 2010.
-  P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price *Access path selection in a relational database management system* In SIGMOD, pages 23–34, 1979.
-  D. P. Woodruff *Optimal space lower bounds for all frequency moments* In Proc. SODA, pages 167–175, 2004.