The Power of Theory in the Practice of Hashing with Focus on Similarity Estimation

Mikkel Thorup

University of Copenhagen

The Power of Theory in the Practice of Hashing with Focus on Similarity Estimation

Mikkel Thorup

University of Copenhagen

Thanks for inviting me to talk here at AAML'24.

Talk surveys results from

- M. Patrascu and M. Thorup: The power of simple tabulation hashing. STOC 2011.
- M. Thorup: Bottom-k and priority sampling, set similarity and subset sums with minimal independence. STOC 2013.
- S. Dahlgaard, M.B.T. Knudsen and E. Rotenberg, and M. Thorup: Hashing for Statistics over K-Partitions. FOCS 2015.
- S. Dahlgaard, M.B.T. Knudsen, M. Thorup: Practical Hash Functions for Similarity Estimation and Dimensionality Reduction. NIPS 2017.
- S. Dahlgaard, M.B.T. Knudsen, M. Thorup: Fast Similarity Sketching. FOCS 2017.
- J.B.T. Houen, and M. Thorup: Understanding the Moments of Tabulation Hashing via Chaoses. ICALP 2022.
- J.B.T. Houen, and M. Thorup: A Sparse Johnson-Lindenstrauss Transform Using Fast Hashing. ICALP 2023.
- I.O. Bercea, L. Beretta, J. Klausen, J.B.T. Houen, and M. Thorup: Locally Uniform Hashing. FOCS 2023:

Generic Target

- Simple and reliable pseudo-random hashing.

Generic Target

- Simple and reliable pseudo-random hashing.
- Providing algorithmically important probabilisitic guarantees akin to those of truly random hashing,

  > *mapping keys to independent uniformly distributed hash values (or hashes),*

  yet easy to implement.

Generic Target

- Simple and reliable pseudo-random hashing.
- Providing algorithmically important probabilisitic guarantees akin to those of truly random hashing,

  > *mapping keys to independent uniformly distributed hash values (or hashes),*

  yet easy to implement.
- Bridging theory (assuming truly random hashing) with practice (needing something implementable).

Generic Target

- Simple and reliable pseudo-random hashing.
- Providing algorithmically important probabilisitic guarantees akin to those of truly random hashing,

  > *mapping keys to independent uniformly distributed hash values (or hashes),*

  yet easy to implement.
- Bridging theory (assuming truly random hashing) with practice (needing something implementable).
- Many randomized algorithms are very simple and popular in practice, but often implemented with too simple hash functions, so guarantees only for sufficiently random input.

Generic Target

- Simple and reliable pseudo-random hashing.
- Providing algorithmically important probabilisitic guarantees akin to those of truly random hashing,

  > *mapping keys to independent uniformly distributed hash values (or hashes),*

  yet easy to implement.
- Bridging theory (assuming truly random hashing) with practice (needing something implementable).
- Many randomized algorithms are very simple and popular in practice, but often implemented with too simple hash functions, so guarantees only for sufficiently random input.
- Too simple hash functions may work deceptively well in random tests, but the real world is full of structured data on which they may fail miserably (as we shall see later).

Wegman & Carter [FOCS'77]

We do not have space for truly random hash functions, but

Family $\mathcal{H} = \{h : [u] \to [r]\}$ *k-independent* iff for rand om $h \in \mathcal{H}$:

- $(\forall) x \in [u]$, $h(x)$ is uniform in $[r]$;
- $(\forall) x_1, \ldots, x_k \in [u]$, $h(x_1), \ldots, h(x_k)$ are independent.

Prototypical example: degree $k-1$ polynomial

- $u = r = p$ prime;
- choose $a_0, a_1, \ldots, a_{k-1}$ uniformly and independently in $[p]$;
- $h(x) = \left(a_0 + a_1 x + \cdots + a_{k-1} x^{k-1}\right) \bmod p$.

For any algorithmic application, we can ask how much independence is needed (but we will do better).

Running Example: Jaccard Set Similarity

- Focus on Jaccard similarity of sets A and B, defined as

$$J(A, B) = |A \cap B| / |A \cup B|.$$

Running Example: Jaccard Set Similarity

- Focus on Jaccard similarity of sets A and B, defined as

$$J(A, B) = |A \cap B| / |A \cup B|.$$

- Construct small sketch $S(A)$ of each set A.

Running Example: Jaccard Set Similarity

- Focus on Jaccard similarity of sets A and B, defined as

$$J(A, B) = |A \cap B| / |A \cup B|.$$

- Construct small sketch $S(A)$ of each set A.
- Use only sketches $S(A)$ and $S(B)$ to estimate $J(A, B)$.

Running Example: Jaccard Set Similarity

- Focus on Jaccard similarity of sets A and B, defined as

$$J(A, B) = |A \cap B| / |A \cup B|.$$

- Construct small sketch $S(A)$ of each set A.
- Use only sketches $S(A)$ and $S(B)$ to estimate $J(A, B)$.
- May have sketches of many sets $S(B_1), \ldots, S(B_n)$: now find B_i similar to A using $S(A)$.

Running Example: Jaccard Set Similarity

- Focus on Jaccard similarity of sets A and B, defined as

$$J(A, B) = |A \cap B| / |A \cup B|.$$

- Construct small sketch $S(A)$ of each set A.
- Use only sketches $S(A)$ and $S(B)$ to estimate $J(A, B)$.
- May have sketches of many sets $S(B_1), \ldots, S(B_n)$: now find B_i similar to A using $S(A)$.
- Hashing used to create comparable sketches.

Running Example: Jaccard Set Similarity

- Focus on Jaccard similarity of sets A and B, defined as

$$J(A, B) = |A \cap B|/|A \cup B|.$$

- Construct small sketch $S(A)$ of each set A.
- Use only sketches $S(A)$ and $S(B)$ to estimate $J(A, B)$.
- May have sketches of many sets $S(B_1), \ldots, S(B_n)$: now find B_i similar to A using $S(A)$.
- Hashing used to create comparable sketches.
- Will study how hash function implementation impacts quality of estimates.

Min-wise hashing [Broder, '98, Alta Vita]

- Jaccard similarity $J(A, B) = |A \cap B| / |A \cup B|$.

Min-wise hashing [Broder, '98, Alta Vita]

- Jaccard similarity $J(A, B) = |A \cap B|/|A \cup B|$.
- Assume truly random collision free hash function h.

Min-wise hashing [Broder, '98, Alta Vita]

- Jaccard similarity $J(A, B) = |A \cap B|/|A \cup B|$.
- Assume truly random collision free hash function h.
- Let $x \in A \cup B$ minimize $h(x)$.

Min-wise hashing [Broder, '98, Alta Vita]

- Jaccard similarity $J(A, B) = |A \cap B|/|A \cup B|$.
- Assume truly random collision free hash function h.
- Let $x \in A \cup B$ minimize $h(x)$.
- Then x uniform in $A \cup B$, so $\Pr[x \in A \cap B] = J(A, B)$.

Min-wise hashing [Broder, '98, Alta Vita]

- Jaccard similarity $J(A, B) = |A \cap B|/|A \cup B|$.
- Assume truly random collision free hash function h.
- Let $x \in A \cup B$ minimize $h(x)$.
- Then x uniform in $A \cup B$, so $\Pr[x \in A \cap B] = J(A, B)$.
- Moreover $x \in A \cap B \iff \min h(A) = \min h(B)$.

Min-wise hashing [Broder, '98, Alta Vita]

- Jaccard similarity $J(A, B) = |A \cap B|/|A \cup B|$.
- Assume truly random collision free hash function h.
- Let $x \in A \cup B$ minimize $h(x)$.
- Then x uniform in $A \cup B$, so $\Pr[x \in A \cap B] = J(A, B)$.
- Moreover $x \in A \cap B \iff \min h(A) = \min h(B)$.
- So $J(A, B) = \Pr_h[\min h(A) = \min h(B)]$.

Min-wise hashing [Broder, ’98, Alta Vita]

- Jaccard similarity $J(A,B) = |A \cap B|/|A \cup B|$.
- Assume truly random collision free hash function h.
- Let $x \in A \cup B$ minimize $h(x)$.
- Then x uniform in $A \cup B$, so $\Pr[x \in A \cap B] = J(A,B)$.
- Moreover $x \in A \cap B \iff \min h(A) = \min h(B)$.
- So $J(A,B) = \Pr_h[\min h(A) = \min h(B)]$.

Thus $\min h(A)$ sketch of A.

Min-wise hashing [Broder, '98, Alta Vita]

- Jaccard similarity $J(A, B) = |A \cap B| / |A \cup B|$.
- Assume truly random collision free hash function h.
- Let $x \in A \cup B$ minimize $h(x)$.
- Then x uniform in $A \cup B$, so $\Pr[x \in A \cap B] = J(A, B)$.
- Moreover $x \in A \cap B \iff \min h(A) = \min h(B)$.
- So $J(A, B) = \Pr_h[\min h(A) = \min h(B)]$.

Thus $\min h(A)$ sketch of A. For concentration, repeat k times:

k-Mins use k independent hash functions $h_1, ..., h_k$
For set A store sketch $M^k(A) = (\min h_1(A), .., \min h_k(A))$.

Min-wise hashing [Broder, '98, Alta Vita]

- Jaccard similarity $J(A, B) = |A \cap B| / |A \cup B|$.
- Assume truly random collision free hash function h.
- Let $x \in A \cup B$ minimize $h(x)$.
- Then x uniform in $A \cup B$, so $\Pr[x \in A \cap B] = J(A, B)$.
- Moreover $x \in A \cap B \iff \min h(A) = \min h(B)$.
- So $J(A, B) = \Pr_h[\min h(A) = \min h(B)]$.

Thus $\min h(A)$ sketch of A. For concentration, repeat k times:

k-Mins use k independent hash functions $h_1, ..., h_k$
For set A store sketch $M^k(A) = (\min h_1(A), .., \min h_k(A))$.

To estimate Jaccard similariy $J(A, B)$ of A and B, use

$$\sum_{i=1}^{k} [\min h_i(A) = \min h_i(B)]/k$$

Min-wise hashing [Broder, '98, Alta Vita]

- Jaccard similarity $J(A, B) = |A \cap B|/|A \cup B|$.
- Assume truly random collision free hash function h.
- Let $x \in A \cup B$ minimize $h(x)$.
- Then x uniform in $A \cup B$, so $\Pr[x \in A \cap B] = J(A, B)$.
- Moreover $x \in A \cap B \iff \min h(A) = \min h(B)$.
- So $J(A, B) = \Pr_h[\min h(A) = \min h(B)]$.

Thus $\min h(A)$ sketch of A. For concentration, repeat k times:

k-Mins use k independent hash functions $h_1, ..., h_k$
For set A store sketch $M^k(A) = (\min h_1(A), .., \min h_k(A))$.

To estimate Jaccard similariy $J(A, B)$ of A and B, use

$$\sum_{i=1}^{k} [\min h_i(A) = \min h_i(B)]/k$$

Expected relative error below $1/\sqrt{J(A, B) \cdot k}$.

Bias issues

We do not have space for truly random hash functions.

Bias issues

We do not have space for truly random hash functions.

We say h is ε-minwise if for any C, $x \in C$,:

$$\Pr_{h \in \mathcal{H}}[h(x) = \min h(C)] = \frac{1 \pm \varepsilon}{|C|}$$

Bias issues

We do not have space for truly random hash functions.

We say h is ε-minwise if for any C, $x \in C$,:

$$\Pr_{h \in \mathcal{H}}[h(x) = \min h(C)] = \frac{1 \pm \varepsilon}{|C|}$$

Such bias ε takes independence $\Theta\left(\lg \frac{1}{\varepsilon}\right)$

[Indyk'99, Patrascu Thorup '10].

Bias issues

We do not have space for truly random hash functions.

We say h is ε-minwise if for any C, $x \in C$,:

$$\Pr_{h \in \mathcal{H}}[h(x) = \min h(C)] = \frac{1 \pm \varepsilon}{|C|}$$

Such bias ε takes independence $\Theta\left(\lg \frac{1}{\varepsilon}\right)$

[Indyk'99, Patrascu Thorup '10].

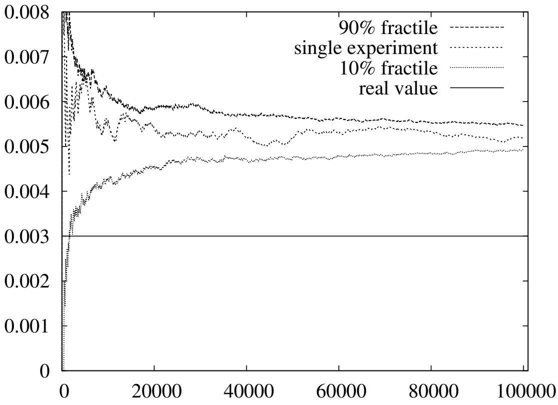
Bias ε not improved by k-mins no matter repetitions k.

Practice with k-mins

Mitzenmacher and Vadhan [SODA'08]: with enough entropy, 2-independent works as good as random,

Practice with k-mins

Mitzenmacher and Vadhan [SODA'08]: with enough entropy, 2-independent works as good as random, but real world full of low-entroy data:



Measures dissimilarity between sets whose intersection are consecutive numbers and the rest is random numbers.

Bottom-k

With single hash function h, the bottom-k sketch of set X is

$$S_k(X) = \{k \text{ elements of } X \text{ with smallest hash values}\}$$

Bottom-k

With single hash function h, the bottom-k sketch of set X is

$$S_k(X) = \{k \text{ elements of } X \text{ with smallest hash values}\}$$

For subset $Y \subseteq X$, estimate frequency $f = |Y|/|X|$ as

$$|S_k(X) \cap Y|/k$$

Bottom-k

With single hash function h, the bottom-k sketch of set X is

$$S_k(X) = \{k \text{ elements of } X \text{ with smallest hash values}\}$$

For subset $Y \subseteq X$, estimate frequency $f = |Y|/|X|$ as

$$|S_k(X) \cap Y|/k$$

For sets A and B, sketch of union $A \cup B$ computed as

$$S_k(A \cup B) = S_k(S_k(A) \cup S_k(B))$$

and Jaccard similarity $J(A, B)$ is estimated as

$$|S_k(A \cup B) \cap S_k(A) \cap S_k(B)|/k$$

Bottom-k

With single hash function h, the bottom-k sketch of set X is

$$S_k(X) = \{k \text{ elements of } X \text{ with smallest hash values}\}$$

For subset $Y \subseteq X$, estimate frequency $f = |Y|/|X|$ as

$$|S_k(X) \cap Y|/k$$

For sets A and B, sketch of union $A \cup B$ computed as

$$S_k(A \cup B) = S_k(S_k(A) \cup S_k(B))$$

and Jaccard similarity $J(A, B)$ is estimated as

$$|S_k(A \cup B) \cap S_k(A) \cap S_k(B)|/k$$

Theorem [T STOC'13] If h is 2-independent, even including bias, the expected relative error is $O(1/\sqrt{f \cdot k})$.

Bottom-k

With single hash function h, the bottom-k sketch of set X is

$$S_k(X) = \{k \text{ elements of } X \text{ with smallest hash values}\}$$

For subset $Y \subseteq X$, estimate frequency $f = |Y|/|X|$ as

$$|S_k(X) \cap Y|/k$$

For sets A and B, sketch of union $A \cup B$ computed as

$$S_k(A \cup B) = S_k(S_k(A) \cup S_k(B))$$

and Jaccard similarity $J(A, B)$ is estimated as

$$|S_k(A \cup B) \cap S_k(A) \cap S_k(B)|/k$$

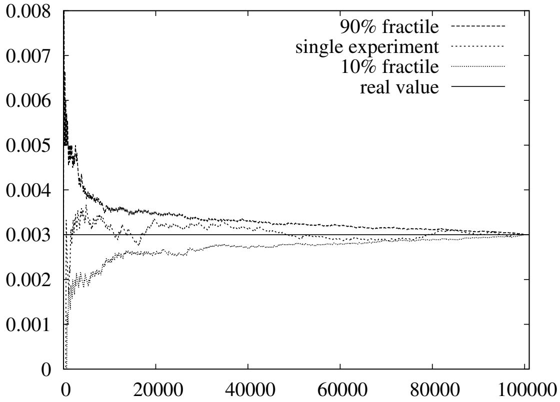
Theorem [T STOC'13] If h is 2-independent, even including bias, the expected relative error is $O(1/\sqrt{f \cdot k})$.

Porat proved this for 8-independent hashing and $k \gg 1$.

Practice with bottom-k

We can use Dietzfelbinger's super fast 2-independent hashing

```
random int64 a,b;
h(int32 x) return (a*x + b) >> 32;
```



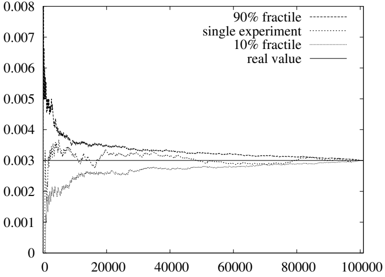
Measures dissimilarity between sets whose intersection are consecutive numbers and the rest is random numbers.

Practice

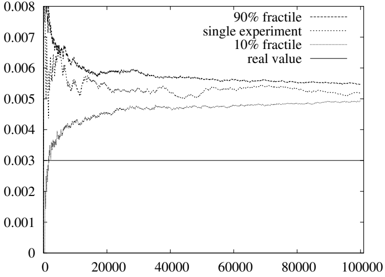
We can use Dietzfelbinger's super fast 2-independent hashing

```
random int64 a,b;
h(int32 x) return (a*x + b) >> 32;
```

bottom-k



k-mins

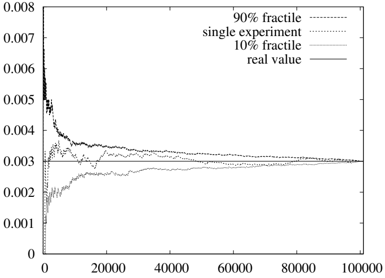


Practice

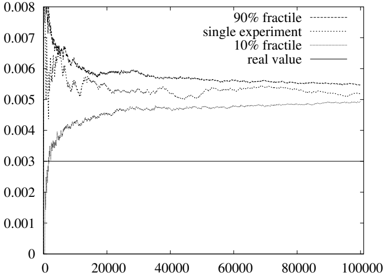
We can use Dietzfelbinger's super fast 2-independent hashing

```
random int64 a,b;
h(int32 x) return (a*x + b) >> 32;
```

bottom-*k*



k-mins



Besides diminishing bias with bottom-*k*, we have stronger convergence because it uses sampling without replacement, whereas *k*-mins is with replacement.

Analysis of Bottom-k

With fully-random hashing, analysis is easy:

General Theorem If we select a set S of k elements from X based on fully random hash values, then S is uniform in

$$\binom{X}{k}.$$

That is, S consists of k uniform samples without replacement.

Analysis of Bottom-k

With fully-random hashing, analysis is easy:

General Theorem If we select a set S of k elements from X based on fully random hash values, then S is uniform in

$$\binom{X}{k}.$$

That is, S consists of k uniform samples without replacement.

Proof If we first fix the set of hash values, then we know the hash values of the k elements selected.

Analysis of Bottom-k

With fully-random hashing, analysis is easy:

General Theorem If we select a set S of k elements from X based on fully random hash values, then S is uniform in

$$\binom{X}{k}.$$

That is, S consists of k uniform samples without replacement.

Proof If we first fix the set of hash values, then we know the hash values of the k elements selected.
With fully-random hashing, it is fully random which element gets which hash value, hence which elements get selected. $\square$

Analysis of Bottom-k

With fully-random hashing, analysis is easy:

General Theorem If we select a set S of k elements from X based on fully random hash values, then S is uniform in

$$\binom{X}{k}.$$

That is, S consists of k uniform samples without replacement.

Proof If we first fix the set of hash values, then we know the hash values of the k elements selected.
With fully-random hashing, it is fully random which element gets which hash value, hence which elements get selected. $\square$

For $Y \subseteq X$, estimate $f = |Y|/|X|$ by $|Y \cap S|/k$.

Analysis of Bottom-k

With fully-random hashing, analysis is easy:

General Theorem If we select a set S of k elements from X based on fully random hash values, then S is uniform in

$$\binom{X}{k}.$$

That is, S consists of k uniform samples without replacement.

Proof If we first fix the set of hash values, then we know the hash values of the k elements selected.
With fully-random hashing, it is fully random which element gets which hash value, hence which elements get selected. $\square$

For $Y \subseteq X$, estimate $f = |Y|/|X|$ by $|Y \cap S|/k$.
Variable $|Y \cap S|$ has mean fk. Chebyshev and Chernoff bounds also hold without replacement (negative correlation helps).

Analysis of Bottom-k

With fully-random hashing, analysis is easy:

General Theorem If we select a set S of k elements from X based on fully random hash values, then S is uniform in

$$\binom{X}{k}.$$

That is, S consists of k uniform samples without replacement.

Proof If we first fix the set of hash values, then we know the hash values of the k elements selected.
With fully-random hashing, it is fully random which element gets which hash value, hence which elements get selected. $\square$

For $Y \subseteq X$, estimate $f = |Y|/|X|$ by $|Y \cap S|/k$.
Variable $|Y \cap S|$ has mean fk. Chebyshev and Chernoff bounds also hold without replacement (negative correlation helps).
By Chebyshev, expected error $O(\sqrt{fk})$.

Analysis of Bottom-k

With fully-random hashing, analysis is easy:

General Theorem If we select a set S of k elements from X based on fully random hash values, then S is uniform in

$$\binom{X}{k}.$$

That is, S consists of k uniform samples without replacement.

Proof If we first fix the set of hash values, then we know the hash values of the k elements selected.
With fully-random hashing, it is fully random which element gets which hash value, hence which elements get selected. $\square$

For $Y \subseteq X$, estimate $f = |Y|/|X|$ by $|Y \cap S|/k$.
Variable $|Y \cap S|$ has mean fk. Chebyshev and Chernoff bounds also hold without replacement (negative correlation helps).
By Chebyshev, expected error $O(\sqrt{fk})$.

Want same bound for bottom-k sample S with 2-independent hashing, but $x \in S$ depends on all hash values.

Analysis of Bottom-k by Simple Union Bound

With $n = |X|$, $S = S_k(X)$, $Y \subseteq X$, estimate $f = |Y|/|X|$ as

$$|Y \cap S|/k$$

Overestimate: For parameters $a < 1$, b, bound probability

$$(*) \qquad |Y \cap S| > \tfrac{1+b}{1-a} f\, k.$$

Analysis of Bottom-k by Simple Union Bound

With $n = |X|$, $S = S_k(X)$, $Y \subseteq X$, estimate $f = |Y|/|X|$ as

$$|Y \cap S|/k$$

Overestimate: For parameters $a < 1$, b, bound probability

$$(*) \qquad |Y \cap S| > \tfrac{1+b}{1-a} f\, k.$$

Define threshold probability (independent of random choices)

$$p = \frac{k}{n\,(1-a)}.$$

Hash values uniform in $(0,1)$ so $\Pr[h(x) < p] = p$.

Analysis of Bottom-k by Simple Union Bound

With $n = |X|$, $S = S_k(X)$, $Y \subseteq X$, estimate $f = |Y|/|X|$ as

$$|Y \cap S|/k$$

Overestimate: For parameters $a < 1$, b, bound probability

$$(*) \qquad |Y \cap S| > \tfrac{1+b}{1-a} f\, k.$$

Define threshold probability (independent of random choices)

$$p = \frac{k}{n\,(1-a)}.$$

Hash values uniform in $(0,1)$ so $\Pr[h(x) < p] = p$.

Proposition The overestimate $(*)$ implies one of

$$\begin{array}{ll} \text{(A)} & |\{x \in X | h(x) < p\}| < k \\ \text{(B)} & |\{x \in Y | h(x) < p\}| > (1+b)\,p|Y|. \end{array}$$

so $\Pr[(*)] \leq \Pr[(A)] + \Pr[(B)]$.

Analysis of Bottom-k by Simple Union Bound

With $n = |X|$, $S = S_k(X)$, $Y \subseteq X$, estimate $f = |Y|/|X|$ as

$$|Y \cap S|/k$$

Overestimate: For parameters $a < 1$, b, bound probability

$$(*) \qquad |Y \cap S| > \tfrac{1+b}{1-a} f\, k.$$

Define threshold probability (independent of random choices)

$$p = \frac{k}{n(1-a)}.$$

Hash values uniform in $(0,1)$ so $\Pr[h(x) < p] = p$.

Proposition The overestimate $(*)$ implies one of

$$\begin{array}{lll} \text{(A)} & |\{x \in X | h(x) < p\}| < k & = (1-a)\mathbf{E}[\cdot] \\ \text{(B)} & |\{x \in Y | h(x) < p\}| > (1+b)\,p|Y|. & = (1+b)\mathbf{E}[\cdot] \end{array}$$

so $\Pr[(*)] \leq \Pr[(A)] + \Pr[(B)]$.

Proof: ¬(A)∧¬(B) $\implies$ ¬($*$)

Suppose (A) is false, that is,

$$|\{x \in X | h(x) < p\}| \geq k.$$

Proof: $\neg$(A)$\wedge\neg$(B) $\implies \neg(*)$

Suppose (A) is false, that is,

$$|\{x \in X | h(x) < p\}| \geq k.$$

Then, since S contains k smallest,

$$\forall x \in S : h(x) < p.$$

so

$$Y \cap S \subseteq \{x \in Y | h(x) < p\}$$

Proof: $\neg$(A)$\wedge\neg$(B) $\implies$ $\neg(*)$

Suppose (A) is false, that is,

$$|\{x \in X | h(x) < p\}| \geq k.$$

Then, since S contains k smallest,

$$\forall x \in S : h(x) < p.$$

so

$$Y \cap S \subseteq \{x \in Y | h(x) < p\}$$

Suppose (B) is also false, that is,

$$|\{x \in Y | h(x) < p\}| < (1 + b)\, p\, |Y|$$

Proof: $\neg$(A)$\wedge\neg$(B) $\implies$ $\neg(*)$

Suppose (A) is false, that is,

$$|\{x \in X | h(x) < p\}| \geq k.$$

Then, since S contains k smallest,

$$\forall x \in S : h(x) < p.$$

so

$$Y \cap S \subseteq \{x \in Y | h(x) < p\}$$

Suppose (B) is also false, that is,

$$|\{x \in Y | h(x) < p\}| < (1 + b)\, p\, |Y|$$

With $p = k/(n\,(1 - a))$ and $|Y| = f\, n$, we get

$$|Y \cap S| \leq |\{x \in Y | h(x) < p\}| < (1 + b)\, p\, |Y| = \frac{1 + b}{1 - a} f\, k$$

so the overestimate $(*)$ did not happen.

2-independence

Proposition With $p = \frac{k}{n\,(1-a)}$, the overestimate

$$(*) \qquad |Y \cap S| > \tfrac{1+b}{1-a}\, f\, k.$$

implies one of

$$\begin{array}{lll} \text{(A)} & |\{x \in X | h(x) < p\}| < k & = (1-a)\mathbf{E}[\cdot] \\ \text{(B)} & |\{x \in Y | h(x) < p\}| > (1+b)\, p |Y| & = (1+b)\mathbf{E}[\cdot] \end{array}$$

so $\Pr[(*)] \leq \Pr[(A)] + \Pr[(B)]$.

2-independence

Proposition With $p = \frac{k}{n(1-a)}$, the overestimate

$$(*) \qquad |Y \cap S| > \tfrac{1+b}{1-a} f k.$$

implies one of

$$\begin{array}{lll} \text{(A)} & |\{x \in X | h(x) < p\}| < k & = (1-a)\mathbf{E}[\cdot] \\ \text{(B)} & |\{x \in Y | h(x) < p\}| > (1+b)\, p |Y| & = (1+b)\mathbf{E}[\cdot] \end{array}$$

so $\Pr[(*)] \leq \Pr[(\mathrm{A})] + \Pr[(\mathrm{B})]$.

Application If h is 2-independent, by Chebyshev,

$$\begin{aligned} \Pr[(A)] &\leq 1/(a^2 k) \\ \Pr[(B)] &\leq 1/(b^2 f\, k) \end{aligned}$$

2-independence

Proposition With $p = \frac{k}{n(1-a)}$, the overestimate

$$(*) \qquad |Y \cap S| > \tfrac{1+b}{1-a} f\,k.$$

implies one of

$$\begin{array}{lll} \text{(A)} & |\{x \in X | h(x) < p\}| < k & = (1-a)\mathbf{E}[\cdot] \\ \text{(B)} & |\{x \in Y | h(x) < p\}| > (1+b)\,p|Y| & = (1+b)\mathbf{E}[\cdot] \end{array}$$

so $\Pr[(*)] \leq \Pr[\text{(A)}] + \Pr[\text{(B)}]$.

Application If h is 2-independent, by Chebyshev,

$$\begin{aligned} \Pr[(A)] &\leq 1/(a^2 k) \\ \Pr[(B)] &\leq 1/(b^2 f\,k) \end{aligned}$$

For any $\varepsilon < 1/3$, appropriate choice of a and b yields

$$\Pr\left[|Y \cap S| > (1+\varepsilon) f\,k\right] \leq \frac{4}{\varepsilon^2 f\,k}.$$

2-independence

Proposition With $p = \frac{k}{n(1-a)}$, the overestimate

$$(*) \qquad |Y \cap S| > \frac{1+b}{1-a} f\,k.$$

implies one of

$$\begin{array}{lll} \text{(A)} & |\{x \in X | h(x) < p\}| < k & = (1-a)\mathbf{E}[\cdot] \\ \text{(B)} & |\{x \in Y | h(x) < p\}| > (1+b)\,p|Y| & = (1+b)\mathbf{E}[\cdot] \end{array}$$

so $\Pr[(*)] \le \Pr[(A)] + \Pr[(B)]$.

Application If h is 2-independent, by Chebyshev,

$$\begin{aligned} \Pr[(A)] &\le 1/(a^2 k) \\ \Pr[(B)] &\le 1/(b^2 f\,k) \end{aligned}$$

For any $\varepsilon < 1/3$, appropriate choice of a and b yields

$$\Pr\left[|Y \cap S| > (1+\varepsilon) f\,k\right] \le \frac{4}{\varepsilon^2 f\,k}.$$

With more calcluations

$$E[||Y \cap S| - f\,k|] = O(\sqrt{f \cdot k}).$$

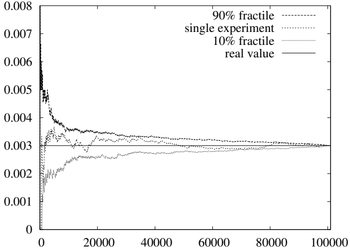
Conclusion on k-mins versus bottom-k

- Both k-mins and bottom-k generalize the idea of storing the smallest hash value.

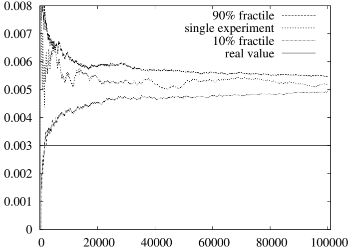
Conclusion on *k*-mins versus bottom-*k*

- Both *k*-mins and bottom-*k* generalize the idea of storing the smallest hash value.
- With limited independence
  k-mins has major problems with bias whereas bottom-*k* works perfectly even with 2-independence.

bottom-*k*



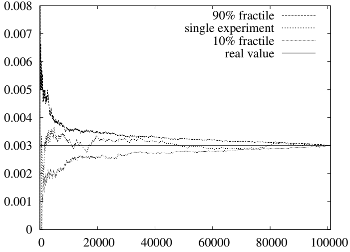
k-mins



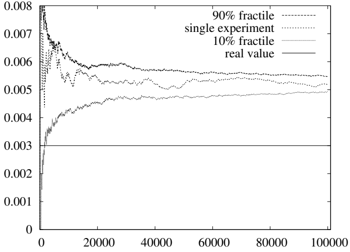
Conclusion on k-mins versus bottom-k

- Both k-mins and bottom-k generalize the idea of storing the smallest hash value.
- With limited independence
  k-mins has major problems with bias whereas bottom-k works perfectly even with 2-independence.

bottom-k



k-mins

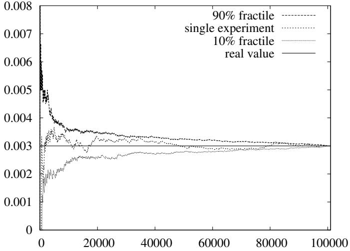


- Analysis: union bound over two simple Chebyshev bounds.

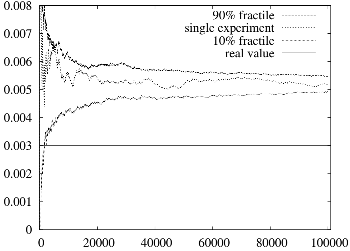
Conclusion on *k*-mins versus bottom-*k*

- Both *k*-mins and bottom-*k* generalize the idea of storing the smallest hash value.
- With limited independence
  k-mins has major problems with bias whereas bottom-*k* works perfectly even with 2-independence.

bottom-*k*



k-mins



- Analysis: union bound over two simple Chebyshev bounds.
- Bottom-*k* also efficient in steaming: maintain *k* smallest elements in priority queue in $O(\log k)$ time per element.

Similarity Vector Sample

- Sketch S maps any set A to vector of k samples from A.

Similarity Vector Sample

- Sketch S maps any set A to vector of k samples from A.
- Want S to preserve similarity between sets A and B:

Similarity Vector Sample

- Sketch S maps any set A to vector of k samples from A.
- Want S to preserve similarity between sets A and B:
- For each coordinate $i \in [k]$, unbiased estimator

$$\Pr[S(A)_i = S(B)_i] = J(A, B) = |A \cap B| / |A \cup B|.$$

Similarity Vector Sample

- Sketch S maps any set A to vector of k samples from A.
- Want S to preserve similarity between sets A and B:
- For each coordinate $i \in [k]$, unbiased estimator

$$\Pr[S(A)_i = S(B)_i] = J(A, B) = |A \cap B| / |A \cup B|.$$

- Let

$$J_S(A, B) = \sum_{i \in [k]} [S(A)_i = S(B)_i] / k.$$

Similarity Vector Sample

- Sketch S maps any set A to vector of k samples from A.
- Want S to preserve similarity between sets A and B:
- For each coordinate $i \in [k]$, unbiased estimator

$$\Pr[S(A)_i = S(B)_i] = J(A, B) = |A \cap B| / |A \cup B|.$$

- Let
$$J_S(A, B) = \sum_{i \in [k]} [S(A)_i = S(B)_i] / k.$$
- Want strong concentration of $J_S(A, B)$ around $J(A, B)$.

Similarity Vector Sample

- Sketch S maps any set A to vector of k samples from A.
- Want S to preserve similarity between sets A and B:
- For each coordinate $i \in [k]$, unbiased estimator

$$\Pr[S(A)_i = S(B)_i] = J(A,B) = |A \cap B|/|A \cup B|.$$

- Let

$$J_S(A,B) = \sum_{i \in [k]} [S(A)_i = S(B)_i]/k.$$

- Want strong concentration of $J_S(A,B)$ around $J(A,B)$.
- The alignment property, that $S(A)_i$ only compared with $S(B)_i$ important, e.g., for Support Vector Machines (SVMs).

Similarity Vector Sample

- Sketch S maps any set A to vector of k samples from A.
- For each coordinate $i \in [k]$, unbiased estimator

$$\Pr[S(A)_i = S(B)_i] = J(A, B) = |A \cap B|/|A \cup B|.$$

- The alignment property, that $S(A)_i$ only compared with $S(B)_i$ needed for Support Vector Machines (SVMs).

Similarity Vector Sample

- Sketch S maps any set A to vector of k samples from A.
- For each coordinate $i \in [k]$, unbiased estimator

$$\Pr[S(A)_i = S(B)_i] = J(A, B) = |A \cap B|/|A \cup B|.$$

- The alignment property, that $S(A)_i$ only compared with $S(B)_i$ needed for Support Vector Machines (SVMs).
- Code similarity as inner product:
  $h_i : U \to \pm\sqrt{1/k}$ and $S_i^h(A) = h_i(S_i(A))$. Then

$$\mathbf{E}[S^h(A) \cdot S^h(B)] = J(A, B).$$

Similarity Vector Sample

- Sketch S maps any set A to vector of k samples from A.
- For each coordinate $i \in [k]$, unbiased estimator

$$\Pr[S(A)_i = S(B)_i] = J(A, B) = |A \cap B| / |A \cup B|.$$

- The alignment property, that $S(A)_i$ only compared with $S(B)_i$ needed for Support Vector Machines (SVMs).
- Code similarity as inner product:
  $h_i : U \to \pm\sqrt{1/k}$ and $S_i^h(A) = h_i(S_i(A))$. Then

$$\mathbf{E}[S^h(A) \cdot S^h(B)] = J(A, B).$$

- Alignment also needed for Locallity Sensitive Hashing (LSH).

Similarity Vector Sample

- Sketch S maps any set A to vector of k samples from A.
- For each coordinate $i \in [k]$, unbiased estimator

$$\Pr[S(A)_i = S(B)_i] = J(A, B) = |A \cap B|/|A \cup B|.$$

- The alignment property, that $S(A)_i$ only compared with $S(B)_i$ needed for Support Vector Machines (SVMs).
- Code similarity as inner product:
  $h_i : U \to \pm\sqrt{1/k}$ and $S_i^h(A) = h_i(S_i(A))$. Then

$$\mathbf{E}[S^h(A) \cdot S^h(B)] = J(A, B).$$

- Alignment also needed for Locallity Sensitive Hashing (LSH).
- Alignment not satisfied by bottom-k sketch.

Similarity Vector Sample

- Sketch S maps any set A to vector of k samples from A.
- Want S to preserve similarity between sets A and B:
- For each coordinate $i \in [k]$, unbiased estimator

$$\Pr[S(A)_i = S(B)_i] = J(A, B) = |A \cap B|/|A \cup B|.$$

- Let

$$J_S(A, B) = \sum_{i \in [k]} [S(A)_i = S(B)_i]/k.$$

- Want strong concentration of $J_S(A, B)$ around $J(A, B)$.

Similarity Vector Sample

- Sketch S maps any set A to vector of k samples from A.
- Want S to preserve similarity between sets A and B:
- For each coordinate $i \in [k]$, unbiased estimator

$$\Pr[S(A)_i = S(B)_i] = J(A, B) = |A \cap B|/|A \cup B|.$$

- Let

$$J_S(A, B) = \sum_{i \in [k]} [S(A)_i = S(B)_i]/k.$$

- Want strong concentration of $J_S(A, B)$ around $J(A, B)$.
- Typical scenario is a large familty of sets $B_1, \ldots, B_n$.
- For query set A, find one of the most similar B_i comparing $S(A)$ with stored sketches $S(B_1), \ldots, S(B_n)$.

Similarity Vector Sample

- Sketch S maps any set A to vector of k samples from A.
- Want S to preserve similarity between sets A and B:
- For each coordinate $i \in [k]$, unbiased estimator

$$\Pr[S(A)_i = S(B)_i] = J(A, B) = |A \cap B|/|A \cup B|.$$

- Let

$$J_S(A, B) = \sum_{i \in [k]} [S(A)_i = S(B)_i]/k.$$

- Want strong concentration of $J_S(A, B)$ around $J(A, B)$.
- Typical scenario is a large familty of sets $B_1, \ldots, B_n$.
- For query set A, find one of the most similar B_i comparing $S(A)$ with stored sketches $S(B_1), \ldots, S(B_n)$.
- No B_i should look similar to A due to noise in $S(B_i)$.

Similarity Vector Sample

- Sketch S maps any set A to vector of k samples from A.
- Want S to preserve similarity between sets A and B:
- For each coordinate $i \in [k]$, unbiased estimator

$$\Pr[S(A)_i = S(B)_i] = J(A, B) = |A \cap B|/|A \cup B|.$$

- Let

$$J_S(A, B) = \sum_{i \in [k]} [S(A)_i = S(B)_i]/k.$$

- Want strong concentration of $J_S(A, B)$ around $J(A, B)$.
- Typical scenario is a large familty of sets $B_1, \ldots, B_n$.
- For query set A, find one of the most similar B_i comparing $S(A)$ with stored sketches $S(B_1), \ldots, S(B_n)$.
- No B_i should look similar to A due to noise in $S(B_i)$.
- With exponential concentration, suffices with $k = \Theta(\log n)$.

Consistency and uniformity

For each $i \in [t]$, we want

- *consistency*:

$$S(C)_i \in A \subseteq C \implies S(A)_i = S(C)_i.$$

- *uniformity*: $S(C)_i$ is uniform in C.

Consistency and uniformity

For each $i \in [t]$, we want

- *consistency*:

$$S(C)_i \in A \subseteq C \implies S(A)_i = S(C)_i.$$

- *uniformity*: $S(C)_i$ is uniform in C.

Consistency implies

$$S(A)_i = S(B)_i \iff S(A \cup B)_i \in A \cap B$$

So Jaccard estimator

$$J_S(A, B) = \sum_{i \in [t]} [S(A)_i = S(B)_i]/t = \text{fraction of } S(A \cup B) \text{ in } A \cap B.$$

Consistency and uniformity

For each $i \in [t]$, we want

- *consistency*:

$$S(C)_i \in A \subseteq C \implies S(A)_i = S(C)_i.$$

- *uniformity*: $S(C)_i$ is uniform in C.

Consistency implies

$$S(A)_i = S(B)_i \iff S(A \cup B)_i \in A \cap B$$

So Jaccard estimator

$$J_S(A, B) = \sum_{i \in [t]} [S(A)_i = S(B)_i]/t = \text{fraction of } S(A \cup B) \text{ in } A \cap B.$$

Each sample $S(A \cup B)_i$ uniform in $A \cup B$, so unbiased:

$$\Pr[S(A)_i = S(B)_i] = \Pr[S(A \cup B)_i \in A \cap B] = |A \cap B|/|A \cup B| = J(A, B).$$

Similarity Vector Sample with Fully Random Hashing

- Will now discuss designs of similarity vector samplees.

Similarity Vector Sample with Fully Random Hashing

- Will now discuss designs of similarity vector samplees.
- For now, assume free access to fully random hash function.

Similarity Vector Sample with Fully Random Hashing

- Will now discuss designs of similarity vector samplees.
- For now, assume free access to fully random hash function.
- Later, replace all the fully random hash functions with a single tornado tabulation hash function.

Similarity Vector Sample with Fully Random Hashing

- Will now discuss designs of similarity vector samplees.
- For now, assume free access to fully random hash function.
- Later, replace all the fully random hash functions with a single tornado tabulation hash function.
- In black box manner (that also applies to other algorithms), argue that performance almost as good as with full randomness.

Similarity Vector Sample with Fully Random Hashing

- Will now discuss designs of similarity vector samplees.
- For now, assume free access to fully random hash function.
- Later, replace all the fully random hash functions with a single tornado tabulation hash function.
- In black box manner (that also applies to other algorithms), argue that performance almost as good as with full randomness.
- Finally, with experiments, we demonstrate significance of using the right hash function.

Vector sample with k-mins of Broder et al.

For each $i \in [k]$ independently,

- we have hash function $h_i : U \to [0, 1)$,
- and define $S(A)_i = \text{argmin}_{x \in A} h_i(x)$.

Example

$k = 4$, $A = \{5, 27, 52, 73, 99\}$.

Ordered by h_i min on top

h_0	5	h_1	73	h_2	52	h_3	73
	99		27		99		5
	27		99		27		52
	52		5		5		99
	73		52		73		27

$S(A) = (5, 73, 52, 73)$.

Vector sample with k-mins of Broder et al.

For each $i \in [k]$ independently,

- we have hash function $h_i : U \to [0, 1)$,
- and define $S(A)_i = \text{argmin}_{x \in A} h_i(x)$.

Properites

- Consistent for if $y \in A \subseteq C$ then

$$y = \text{argmin}_{x \in C} h_i(x) \implies y = \text{argmin}_{x \in A} h_i(x).$$

Vector sample with k-mins of Broder et al.

For each $i \in [k]$ independently,

- we have hash function $h_i : U \to [0, 1)$,
- and define $S(A)_i = \text{argmin}_{x \in A} h_i(x)$.

Properites

- Consistent for if $y \in A \subseteq C$ then

$$y = \text{argmin}_{x \in C} h_i(x) \implies y = \text{argmin}_{x \in A} h_i(x).$$

- also uniform if h_i *min-wise*, i.e., if $x \in C$,

$$\Pr[h_i(x) = \min h_i(C)] = 1/|C|.$$

Vector sample with k-mins of Broder et al.

For each $i \in [k]$ independently,

- we have hash function $h_i : U \to [0, 1)$,
- and define $S(A)_i = \text{argmin}_{x \in A} h_i(x)$.

Properites

- Consistent for if $y \in A \subseteq C$ then
$$y = \text{argmin}_{x \in C} h_i(x) \implies y = \text{argmin}_{x \in A} h_i(x).$$
- also uniform if h_i *min-wise*, i.e., if $x \in C$,
$$\Pr[h_i(x) = \min h_i(C)] = 1/|C|.$$
- The k samples in $S(A \cup B)$ are independent, so frequency $J_S(A, B)$ of $S(A \cup B)$ in $A \cap B$ strongly concentrated around mean $J(A, B)$. e.g., by Chernoff bounds.

Vector Sample with k-mins of Broder et al.

For each $i \in [k]$,

- we have independent hash function $h_i : U \to [0, 1)$.
- Then we define $S(A)_i = \text{argmin}_{x \in A} h_i(x)$.

We get both unbiased and strongly concentrated similarity estimates, but computing $S(A)$ takes $O(|A| \cdot k)$ time.

Vector Sample with k-mins of Broder et al.

For each $i \in [k]$,

- we have independent hash function $h_i : U \to [0, 1)$.
- Then we define $S(A)_i = \text{argmin}_{x \in A} h_i(x)$.

We get both unbiased and strongly concentrated similarity estimates, but computing $S(A)$ takes $O(|A| \cdot k)$ time.

[FOCS'17] Fast similarity vector sample which is also unbiased and strongly concentrated, computed in $O(|A| + k \log k)$ time.

Vector Sample with k-mins of Broder et al.

For each $i \in [k]$,

- we have independent hash function $h_i : U \to [0, 1)$.
- Then we define $S(A)_i = \text{argmin}_{x \in A} h_i(x)$.

We get both unbiased and strongly concentrated similarity estimates, but computing $S(A)$ takes $O(|A| \cdot k)$ time.

[FOCS'17] Fast similarity vector sample which is also unbiased and strongly concentrated, computed in $O(|A| + k \log k)$ time.

- Li [KDD'12] uses $k > 1000$—for error ε, we need $k = \Omega((\log n)/\varepsilon^2)$

Vector Sample with k-mins of Broder et al.

For each $i \in [k]$,

- we have independent hash function $h_i : U \to [0, 1)$.
- Then we define $S(A)_i = \text{argmin}_{x \in A} h_i(x)$.

We get both unbiased and strongly concentrated similarity estimates, but computing $S(A)$ takes $O(|A| \cdot k)$ time.

[FOCS'17] Fast similarity vector sample which is also unbiased and strongly concentrated, computed in $O(|A| + k \log k)$ time.

- Li [KDD'12] uses $k > 1000$—for error ε, we need $k = \Omega((\log n)/\varepsilon^2)$
- Bachrach and Porat [ICALP'13] for certain pairwise independent polynomials h_i, compute $S(A)$ in $O(|A| \log k)$ time, but problems with both bias and concentration.

One-Permutation Sampling by Li et al. [NIPS'12]

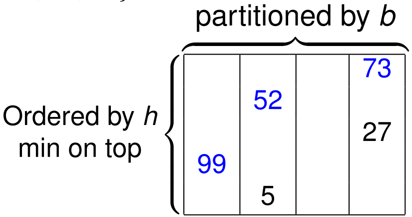
- Two random hash function $h : U \rightarrow [0, 1)$ and $b : U \rightarrow [k]$.
- If there is an $x \in A$ with $b(x) = i$, then

$$S(A)_i = \operatorname{argmin}_{x \in A, b(x)=i} h(x).$$

- Otherwise $S(A)_i$ is undefined.

Example

$k = 4$, $A = \{5, 27, 52, 73, 99\}$.



$S(A) = (99, 52, \div, 73)$.

One-Permutation Sampling by Li et al. [NIPS'12]

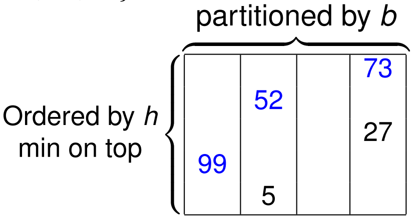
- Two random hash function $h : U \to [0, 1)$ and $b : U \to [k]$.
- If there is an $x \in A$ with $b(x) = i$, then

$$S(A)_i = \text{argmin}_{x \in A, b(x)=i} h(x).$$

- Otherwise $S(A)_i$ is undefined.

Example

$k = 4$, $A = \{5, 27, 52, 73, 99\}$.



$S(A) = (99, 52, \div, 73)$.

- Computed in $O(|A|)$ time.

One-Permutation Sampling by Li et al. [NIPS'12]

- Two random hash function $h : U \to [0,1)$ and $b : U \to [k]$.
- If there is an $x \in A$ with $b(x) = i$, then

$$S(A)_i = \text{argmin}_{x \in A, b(x)=i} h(x).$$

- Otherwise $S(A)_i$ is undefined.
- Computed in $O(|A|)$ time.

Assume $|A \cup B| = \omega(k \log k)$. Then, w.h.p.,

- all $S(A \cup B)_i$ defined, representing
  k samples *without replacement* from $A \cup B$.

One-Permutation Sampling by Li et al. [NIPS'12]

- Two random hash function $h : U \to [0, 1)$ and $b : U \to [k]$.
- If there is an $x \in A$ with $b(x) = i$, then

$$S(A)_i = \mathrm{argmin}_{x \in A, b(x)=i} h(x).$$

- Otherwise $S(A)_i$ is undefined.
- Computed in $O(|A|)$ time.

Assume $|A \cup B| = \omega(k \log k)$. Then, w.h.p.,

- all $S(A \cup B)_i$ defined, representing
  k samples *without replacement* from $A \cup B$.
- Consistent and uniform with Chernoff bounds on fraction $J_S(A, B)$ of $S(A \cup B)$ in $A \cap B$ around mean $J(A, B)$.

One-Permutation Sampling by Li et al. [NIPS'12]

- Two random hash function $h : U \to [0,1)$ and $b : U \to [k]$.
- If there is an $x \in A$ with $b(x) = i$, then

$$S(A)_i = \text{argmin}_{x \in A, b(x)=i} h(x).$$

- Otherwise $S(A)_i$ is undefined.
- Computed in $O(|A|)$ time.

Assume $|A \cup B| = \omega(k \log k)$. Then, w.h.p.,

- all $S(A \cup B)_i$ defined, representing k samples *without replacement* from $A \cup B$.
- Consistent and uniform with Chernoff bounds on fraction $J_S(A,B)$ of $S(A \cup B)$ in $A \cap B$ around mean $J(A,B)$.

But undefined samples pose problems.

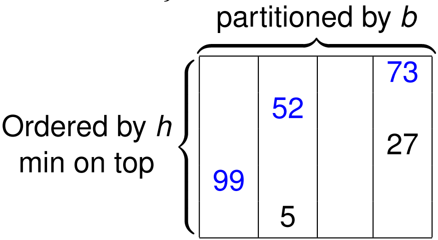
Want sketching procedure with fixed k, not knowing set sizes.

One-permutation sampling with Densification [Shrivastava Li ICML'14,UAI'14,ICML'17]

One-permutation sampling copying defined entries to undefined entries, e.g., copy from left [ICML'14].

Example

$k = 4$, $A = \{5, 27, 52, 73, 99\}$.



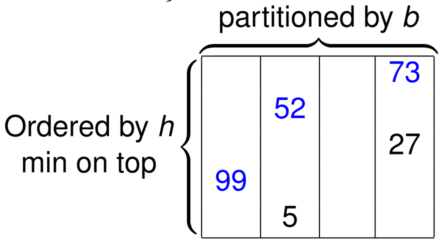
$$S(A) = (99, 52, \div, 73)$$
$$S'(A) = (99, 52, 52, 73).$$

One-permutation sampling with Densification [Shrivastava Li ICML'14,UAI'14,ICML'17]

One-permutation sampling copying defined entries to undefined entries, e.g., copy from left [ICML'14].

Example

$k = 4$, $A = \{5, 27, 52, 73, 99\}$.



$$S(A) = (99, 52, \div, 73)$$
$$S'(A) = (99, 52, 52, 73).$$

but concentration no longer exponential in k.

Fast Similarity Sketch [Dahlgaard, Knudsen, T, FOCS'17]

We have

- k independent hash functions $h^j : U \to [0, 1)$, $j \in [k]$,
- k independent hash function $b^j : U \to [k]$, $j \in [k]$.
- k independent hash function $h_i : U \to [0, 1)$, $i \in [k]$,

Fast Similarity Sketch [Dahlgaard, Knudsen, T, FOCS'17]

We have

- k independent hash functions $h^j : U \to [0,1)$, $j \in [k]$,
- k independent hash function $b^j : U \to [k]$, $j \in [k]$.
- k independent hash function $h_i : U \to [0,1)$, $i \in [k]$,

To compute $S(A)$:

- Initialize $S(A)$ with all empty entries $S(A)_i$.
- For $j = 0, \ldots, t-1$,
  - Make a one-permutation sketch $S^j(A)$ using h^j and b^j.
  - Use defined entries $S^j(A)_i$ to fill empty $S(A)_i$.

Fast Similarity Sketch [Dahlgaard, Knudsen, T, FOCS'17]

We have

- k independent hash functions $h^j : U \to [0, 1)$, $j \in [k]$,
- k independent hash function $b^j : U \to [k]$, $j \in [k]$.
- k independent hash function $h_i : U \to [0, 1)$, $i \in [k]$,

To compute $S(A)$:

- Initialize $S(A)$ with all empty entries $S(A)_i$.
- For $j = 0, \ldots, t - 1$,
  - Make a one-permutation sketch $S^j(A)$ using h^j and b^j.
  - Use defined entries $S^j(A)_i$ to fill empty $S(A)_i$.
  - If all entries of $S(A)_i$ full, return $S(A)$.

Fast Similarity Sketch [Dahlgaard, Knudsen, T, FOCS'17]

We have

- k independent hash functions $h^j : U \to [0, 1)$, $j \in [k]$,
- k independent hash function $b^j : U \to [k]$, $j \in [k]$.
- k independent hash function $h_i : U \to [0, 1)$, $i \in [k]$,

To compute $S(A)$:

- Initialize $S(A)$ with all empty entries $S(A)_i$.
- For $j = 0, \ldots, t - 1$,
  - Make a one-permutation sketch $S^j(A)$ using h^j and b^j.
  - Use defined entries $S^j(A)_i$ to fill empty $S(A)_i$.
  - If all entries of $S(A)_i$ full, return $S(A)$.
- If any empty entries, use last k hash functions for repeated min-wise sketch to fill remaining empty entries, that is, if $S(A)_i$ is empty, set $S(A)_i = \text{argmin}_{x \in A} h_i(x)$.

Example

$k = 4$, $A = \{5, 27, 52, 73, 99\}$.

				73
		52		
One-Permutation				27
	99			
		5		
		52		
			99	
One-Permutation			5	
		27		
				73
Another $k-2$ permutations	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	5	73	52	73
	99	27	99	5
k-Mins	27	99	27	52
	52	5	5	99
	73	52	73	27

$S(A) = (99, 52, 99, 73)$.

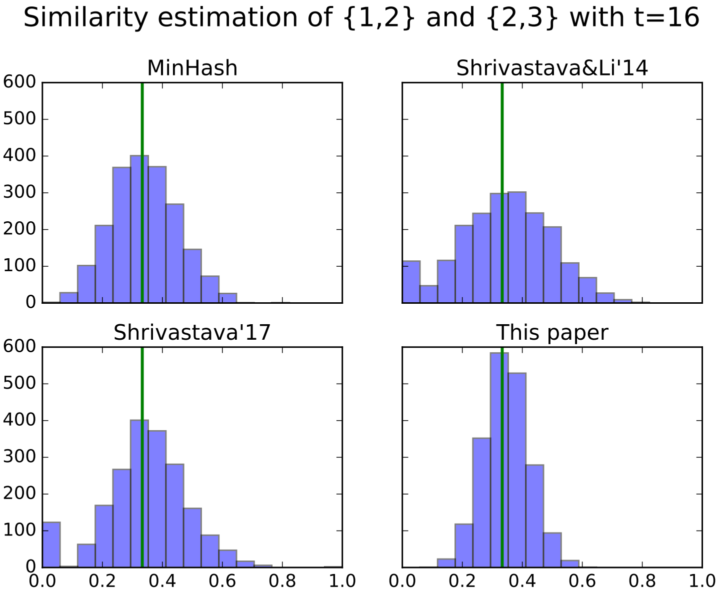
Example

$k = 4$, $A = \{5, 27, 52, 73, 99\}$.

				73
		52		
One-Permutation				27
	99			
		5		
		52		
			99	
One-Permutation			5	
All k samples found,		27		
so we can stop				73
Another $k-2$ permutations	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	5	73	52	73
	99	27	99	5
k-Mins	27	99	27	52
	52	5	5	99
	73	52	73	27

$S(A) = (99, 52, 99, 73)$.

Experiments



Example

$k = 4$, $A = \{5, 27, 52, 73, 99\}$.

				73
		52		
One-Permutation				27
	99			
		5		
		52		
			99	
One-Permutation			5	
All k samples found,		27		
so we can stop				73
Another $k-2$ permutations	⋮	⋮	⋮	⋮
	5	73	52	73
	99	27	99	5
k-Mins	27	99	27	52
	52	5	5	99
	73	52	73	27

Example

$k = 4$, $A = \{5, 27, 52, 73, 99\}$.

				73
		52		
One-Permutation				27
	99			
		5		
		52		
			99	
One-Permutation			5	
All k samples found,		27		
so we can stop				73
Another $k-2$ permutations	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	5	73	52	73
	99	27	99	5
k-Mins	27	99	27	52
	52	5	5	99
	73	52	73	27

Computed in $O(|A| + k \log k)$ w.h.p.

Efficiency

Lemma Our fast similarity sketch computed in $O(|A| + k \log k)$ time, w.h.p.

- Each one-permutation round throws $|A|$ balls into k bins and takes $O(|A|)$ time.

Efficiency

Lemma Our fast similarity sketch computed in $O(|A| + k \log k)$ time, w.h.p.

- Each one-permutation round throws $|A|$ balls into k bins and takes $O(|A|)$ time.
- All bins full, w.h.p., when $\omega(k \log k)$ balls thrown.

Efficiency

Lemma Our fast similarity sketch computed in $O(|A| + k \log k)$ time, w.h.p.

- Each one-permutation round throws $|A|$ balls into k bins and takes $O(|A|)$ time.
- All bins full, w.h.p., when $\omega(k \log k)$ balls thrown.
- E.g., if $|A| = \omega(k \log k)$, w.h.p., all full after first one-permutation sketch.

Efficiency

Lemma Our fast similarity sketch computed in $O(|A| + k \log k)$ time, w.h.p.

- Each one-permutation round throws $|A|$ balls into k bins and takes $O(|A|)$ time.
- All bins full, w.h.p., when $\omega(k \log k)$ balls thrown.
- E.g., if $|A| = \omega(k \log k)$, w.h.p., all full after first one-permutation sketch.
- If we end up in the final k-Mins sketch, then this only doubles the total time.

With and without replacement

- With classic repeated min-wise hashing, we get k independent samples with replacements from $A \cup B$. Takes $O(|A| \cdot k)$ time

With and without replacement

- With classic repeated min-wise hashing, we get k independent samples with replacements from $A \cup B$. Takes $O(|A| \cdot k)$ time
- With one-permutation sampling, we get $t' \leq k$ samples without replacement from $A \cup B$. Sampling without replacement is gives as good a concentration on the fraction of elements from $A \cap B$. Takes $O(|A|)$ time, but problems with undefined entries when $t' < k$.

With and without replacement

- With classic repeated min-wise hashing, we get k independent samples with replacements from $A \cup B$. Takes $O(|A| \cdot k)$ time
- With one-permutation sampling, we get $t' \leq k$ samples without replacement from $A \cup B$. Sampling without replacement is gives as good a concentration on the fraction of elements from $A \cap B$. Takes $O(|A|)$ time, but problems with undefined entries when $t' < k$.
- Fast similarity sketch mixes with and without replacement:
  - We repeat rounds of one-permutation sampling.
  - Round j adds some samples without replacement, but may repeat samples from previous rounds.
  - The final repeated min-wise hashing do remaining samples with replacement.

With and without replacement

- With classic repeated min-wise hashing, we get k independent samples with replacements from $A \cup B$. Takes $O(|A| \cdot k)$ time
- With one-permutation sampling, we get $t' \leq k$ samples without replacement from $A \cup B$. Sampling without replacement is gives as good a concentration on the fraction of elements from $A \cap B$. Takes $O(|A|)$ time, but problems with undefined entries when $t' < k$.
- Fast similarity sketch mixes with and without replacement:
  - We repeat rounds of one-permutation sampling.
  - Round j adds some samples without replacement, but may repeat samples from previous rounds.
  - The final repeated min-wise hashing do remaining samples with replacement.

  Takes $O(|A| + k \log k)$ time w.h.p.

Chernoff Bound

- Similarity estimator $J_S(A, B)$ unbiased and concentrated around its mean $J = J(A, B)$. For $\delta > 0$,

$$\Pr[J_S(A, B) \geq J(1 + \delta)] \leq \left(\frac{e^{\delta}}{(1 + \delta)^{1+\delta}} \right)^{J\,k},$$

$$\Pr[J_S(A, B) \leq J(1 - \delta)] \leq \left(\frac{e^{-\delta}}{(1 - \delta)^{1-\delta}} \right)^{J\,k}.$$

Chernoff Bound

- Similarity estimator $J_S(A,B)$ unbiased and concentrated around its mean $J = J(A,B)$. For $\delta > 0$,

$$\Pr[J_S(A,B) \geq J(1+\delta)] \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{Jk},$$
$$\Pr[J_S(A,B) \leq J(1-\delta)] \leq \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{Jk}.$$

Proof idea Negative correlation from without replacement can only decrease the moments in standard proof based on Taylor expansion.

Using Realistic Hash Functions

- Will now discuss implementation of fast similarity sketching with realistic hash functions.

Using Realistic Hash Functions

- Will now discuss implementation of fast similarity sketching with realistic hash functions.
- For now, we assume that sets has $\omega(k \log k)$ elements.

Using Realistic Hash Functions

- Will now discuss implementation of fast similarity sketching with realistic hash functions.
- For now, we assume that sets has $\omega(k \log k)$ elements.
- Then fast similarity sketching same as one-permutation sampling, since all k samples found, w.h.p.

Using Realistic Hash Functions

- Will now discuss implementation of fast similarity sketching with realistic hash functions.
- For now, we assume that sets has $\omega(k \log k)$ elements.
- Then fast similarity sketching same as one-permutation sampling, since all k samples found, w.h.p.

Example $k = 4$, $A = \{5, 27, 52, 73, 99, 63, 96\}$.

	96			73
		52		
One-Permutation			63	27
All k samples found,	99			
so we can stop		5		
		52		96
			99	
One-Permutation	63		5	
		27		
				73
Another $k-2$ permutations	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	5	73	52	73

Using Realistic Hash Functions

- Will now discuss implementation of fast similarity sketching with realistic hash functions.
- For now, we assume that sets has $\omega(k \log k)$ elements.
- Then fast similarity sketching same as one-permutation sampling, since all k samples found, w.h.p.

Using Realistic Hash Functions

- Will now discuss implementation of fast similarity sketching with realistic hash functions.
- For now, we assume that sets has $\omega(k \log k)$ elements.
- Then fast similarity sketching same as one-permutation sampling, since all k samples found, w.h.p.
- set $R = A \cap B$ of red balls and
- set $B = (A \cup B) \setminus R$ of blue balls.

Using Realistic Hash Functions

- Will now discuss implementation of fast similarity sketching with realistic hash functions.
- For now, we assume that sets has $\omega(k \log k)$ elements.
- Then fast similarity sketching same as one-permutation sampling, since all k samples found, w.h.p.
- set $R = A \cap B$ of red balls and
- set $B = (A \cup B) \setminus R$ of blue balls.
- We make a vector S of k samples from $R \cup B$.

Using Realistic Hash Functions

- Will now discuss implementation of fast similarity sketching with realistic hash functions.
- For now, we assume that sets has $\omega(k \log k)$ elements.
- Then fast similarity sketching same as one-permutation sampling, since all k samples found, w.h.p.
- set $R = A \cap B$ of red balls and
- set $B = (A \cup B) \setminus R$ of blue balls.
- We make a vector S of k samples from $R \cup B$.
- Want number of red samples $|R \cap S|$ concentrated around expectation $k|R|/|R \cup B|$.

Implementing one-permuation hashing

Generic implementation of one-permutation sampling uses single hash function $H : U \to [2^\ell]$

$$H(x) = \overbrace{0010101110010101001}^{\substack{\text{bucket} \\ b(x) = \text{first } \log_2 k \text{ bits}}}\overbrace{00011001010001110010011100}^{\substack{\text{local hash} \\ h(x) = \text{remaining } \ell - \log_2 k \text{ bits}}}$$

$$S(A)_i = \text{argmin}_{x \in A, b(x) = i} h(x).$$

Example

$k = 4$, $A = \{5, 27, 52, 73, 99, 63, 96\}$.

partitioned by b

Ordered by h
min on top

96			73
	52		
		63	27
99			
	5		

$S(A) = (96, 52, 63, 73)$.

Too simple implementation of one-permuation hashing

Classic $H(x) = (ax + b) \bmod p$, Mersenne prime $p = 2^\ell - 1$.

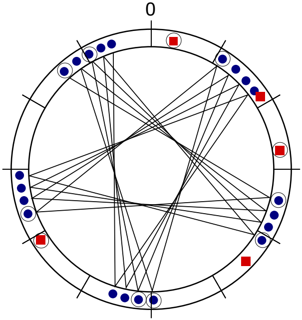
$$H(x) = \overbrace{0010101110010101001}^{b(x)\,=\,\text{first }\log_2 k\text{ bits}}\overbrace{0001100101000111001001 1100}^{h(x)\,=\,\text{remaining }\ell-\log_2 k\text{ bits}}$$

Too simple implementation of one-permuation hashing

Classic $H(x) = (ax + b) \bmod p$, Mersenne prime $p = 2^\ell - 1$.

$$H(x) = \overbrace{0010101110010101001}^{b(x) = \text{first } \log_2 k \text{ bits}}\overbrace{0001100101000111001001 1100}^{h(x) = \text{remaining } \ell - \log_2 k \text{ bits}}$$

$B = \{1, \ldots, 25\}$ (consecutive) and
$R = \{926, 53, 478, 734, 263\}$ (random outliers)

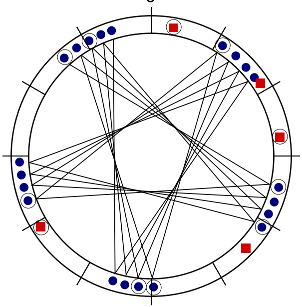


Too simple implementation of one-permuation hashing

Classic $H(x) = (ax + b) \bmod p$, Mersenne prime $p = 2^\ell - 1$.

$$H(x) = \overbrace{0010101110010101001}^{b(x) = \text{first } \log_2 k \text{ bits}}\overbrace{00011001010001110010011100}^{h(x) = \text{remaining } \ell - \log_2 k \text{ bits}}$$

$B = \{1, \ldots, 25\}$ (consecutive) and
$R = \{926, 53, 478, 734, 263\}$ (random outliers)



$S(R \cap B) = (■, •, ■, •, •, •, •, ■, •, \div, •)$.
Then $|S \cap R|/|S| = 3/11$ far from $|R|/|R \cup B| = 5/30 = 1/6$.

Need more independence between buckets

- one-permutation sampling throws elements into k buckets, and pick a random element from each bucket.
- k samples without replacement if all buckets non-empty,
- Sampling from $R \cup B$.
- With full randomness, fraction of samples from R concentrated around $|R|/|R \cup B|$ with Chernoff bounds —without replacement only decreases the moments.

Need more independence between buckets

- one-permutation sampling throws elements into k buckets, and pick a random element from each bucket.
- k samples without replacement if all buckets non-empty,
- Sampling from $R \cup B$.
- With full randomness, fraction of samples from R concentrated around $|R|/|R \cup B|$ with Chernoff bounds —without replacement only decreases the moments.
- Want implementable hash function yielding similar concentration.
- Requires high indpendence between bucket contents —not clear if k independence suffices.

Need more independence between buckets

- one-permutation sampling throws elements into k buckets, and pick a random element from each bucket.
- k samples without replacement if all buckets non-empty,
- Sampling from $R \cup B$.
- With full randomness, fraction of samples from R concentrated around $|R|/|R \cup B|$ with Chernoff bounds —without replacement only decreases the moments.
- Want implementable hash function yielding similar concentration.
- Requires high indpendence between bucket contents —not clear if k independence suffices.
- This is a very generic issue: using hashing to distribute elements in buckets, and aggregating statistics over the buckets to get well-concentrated estimates.
- Idea also used for counting distinct elements [Flajolet et al. FOCS'83], count sketches, and feature hashing.

Need more independence between buckets

- One-Permutation Sampling throws elements into k buckets, and pick a random element from each bucket.
- k samples without replacement if all buckets non-empty,
- Sampling from $R \cup B$.
- With full randomness, fraction of samples from R concentrated around $|R|/|R \cup B|$ with Chernoff bounds.
- Want implementable hash function yielding similar concentration bounds.

Need more independence between buckets

- One-Permutation Sampling throws elements into k buckets, and pick a random element from each bucket.
- k samples without replacement if all buckets non-empty,
- Sampling from $R \cup B$.
- With full randomness, fraction of samples from R concentrated around $|R|/|R \cup B|$ with Chernoff bounds.
- Want implementable hash function yielding similar concentration bounds.
- We use Tornado Tabulation Hashing which is in black-box fashion is almost as good as fully-random.

Need more independence between buckets

- One-Permutation Sampling throws elements into k buckets, and pick a random element from each bucket.
- k samples without replacement if all buckets non-empty,
- Sampling from $R \cup B$.
- With full randomness, fraction of samples from R concentrated around $|R|/|R \cup B|$ with Chernoff bounds.
- Want implementable hash function yielding similar concentration bounds.
- We use Tornado Tabulation Hashing which is in black-box fashion is almost as good as fully-random.
  - First we review Simple Tabulation.

Need more independence between buckets

- One-Permutation Sampling throws elements into k buckets, and pick a random element from each bucket.
- k samples without replacement if all buckets non-empty,
- Sampling from $R \cup B$.
- With full randomness, fraction of samples from R concentrated around $|R|/|R \cup B|$ with Chernoff bounds.
- Want implementable hash function yielding similar concentration bounds.
- We use Tornado Tabulation Hashing which is in black-box fashion is almost as good as fully-random.
  - First we review Simple Tabulation.
  - Using simple tabulation, we define Mixed Tabulation.

Need more independence between buckets

- One-Permutation Sampling throws elements into k buckets, and pick a random element from each bucket.
- k samples without replacement if all buckets non-empty,
- Sampling from $R \cup B$.
- With full randomness, fraction of samples from R concentrated around $|R|/|R \cup B|$ with Chernoff bounds.
- Want implementable hash function yielding similar concentration bounds.
- We use Tornado Tabulation Hashing which is in black-box fashion is almost as good as fully-random.
  - First we review Simple Tabulation.
  - Using simple tabulation, we define Mixed Tabulation.
  - Then we apply it to One-Permutation Sampling.

Need more independence between buckets

- One-Permutation Sampling throws elements into k buckets, and pick a random element from each bucket.
- k samples without replacement if all buckets non-empty,
- Sampling from $R \cup B$.
- With full randomness, fraction of samples from R concentrated around $|R|/|R \cup B|$ with Chernoff bounds.
- Want implementable hash function yielding similar concentration bounds.
- We use Tornado Tabulation Hashing which is in black-box fashion is almost as good as fully-random.
  - First we review Simple Tabulation.
  - Using simple tabulation, we define Mixed Tabulation.
  - Then we apply it to One-Permutation Sampling.
  - Finally, we show experiments.

Simple Tabulation Hashing [Zobrist'70 chess]

- Key x divided into $c = O(1)$ characters $x_1, ..., x_c \in \Sigma$, e.g., 32-bit key as $4 \times$ 8-bit characters.

Simple Tabulation Hashing [Zobrist'70 chess]

- Key x divided into $c = O(1)$ characters $x_1, ..., x_c \in \Sigma$, e.g., 32-bit key as $4 \times$ 8-bit characters.
- For $i = 1, ..., c$, we have truly random hash table:
  $R_i : \Sigma \to$ hash values (ℓ-bit strings)

Simple Tabulation Hashing [Zobrist'70 chess]

- Key x divided into $c = O(1)$ characters $x_1, ..., x_c \in \Sigma$, e.g., 32-bit key as $4 \times$ 8-bit characters.
- For $i = 1, ..., c$, we have truly random hash table:
  $R_i : \Sigma \to$ hash values (ℓ-bit strings)
- Hash value

$$H(x) = R_1[x_1] \oplus \cdots \oplus R_c[x_c]$$

Simple Tabulation Hashing [Zobrist'70 chess]

- Key x divided into $c = O(1)$ characters $x_1, ..., x_c \in \Sigma$, e.g., 32-bit key as $4 \times$ 8-bit characters.
- For $i = 1, ..., c$, we have truly random hash table:
  $R_i : \Sigma \to$ hash values (ℓ-bit strings)
- Hash value

$$H(x) = R_1[x_1] \oplus \cdots \oplus R_c[x_c]$$

- Space $c|U|^{1/c}$ and time $O(c)$. With 8-bit characters, each H_i has 256 entries and fit in L1 cache.

Simple Tabulation Hashing [Zobrist'70 chess]

- Key x divided into $c = O(1)$ characters $x_1, ..., x_c \in \Sigma$, e.g., 32-bit key as 4×8-bit characters.
- For $i = 1, ..., c$, we have truly random hash table:
  $R_i : \Sigma \to$ hash values (ℓ-bit strings)
- Hash value

$$H(x) = R_1[x_1] \oplus \cdots \oplus R_c[x_c]$$

- Space $c|U|^{1/c}$ and time $O(c)$. With 8-bit characters, each H_i has 256 entries and fit in L1 cache.
- Simple tabulation is the fastest 3-independent hashing scheme. Speed like 2 multiplications.

Simple Tabulation Hashing [Zobrist'70 chess]

- Key x divided into $c = O(1)$ characters $x_1, ..., x_c \in \Sigma$, e.g., 32-bit key as $4 \times$ 8-bit characters.
- For $i = 1, ..., c$, we have truly random hash table:
  $R_i : \Sigma \to$ hash values (ℓ-bit strings)
- Hash value

$$H(x) = R_1[x_1] \oplus \cdots \oplus R_c[x_c]$$

- Space $c|U|^{1/c}$ and time $O(c)$. With 8-bit characters, each H_i has 256 entries and fit in L1 cache.
- Simple tabulation is the fastest 3-independent hashing scheme. Speed like 2 multiplications.
- ... and it has a lot of power..

How much independence needed?

Chaining $\mathbf{E}[t] = O(1)$ $\mathbf{E}[t^k] = O(1)$ $\max t = O(\lg n / \lg\lg n)$	2 $2k+1$ $\Theta(\lg n / \lg\lg n)$	
Linear probing	≤ 5 [Pagh², Ružić'07]	≥ 5 **[PT ICALP'10]**
Cuckoo hashing	$O(\lg n)$	≥ 6 [Cohen, Kane'05]
F_2 estimation	4 [Alon, Mathias, Szegedy'99]	
ε-minwise indep.	$O(\lg \frac{1}{\varepsilon})$ [Indyk'99]	$\Omega(\lg \frac{1}{\varepsilon})$ **[PT ICALP'10]**

How much independence needed? Wrong question

Chaining $\mathbf{E}[t] = O(1)$ $\mathbf{E}[t^k] = O(1)$ $\max t = O(\lg n / \lg\lg n)$	2 $2k+1$ $\Theta(\lg n / \lg\lg n)$	
Linear probing	≤ 5 [Pagh², Ružić'07]	≥ 5 **[PT ICALP'10]**
Cuckoo hashing	$O(\lg n)$	≥ 6 [Cohen, Kane'05]
F_2 estimation	4 [Alon, Mathias, Szegedy'99]	
ε-minwise indep.	$O(\lg \frac{1}{\varepsilon})$ [Indyk'99]	$\Omega(\lg \frac{1}{\varepsilon})$ **[PT ICALP'10]**

Patrascu-T'11: Despite its 4-dependence, simple tabulation suffices for all the above applications:

One simple and fast hashing scheme for many needs.

How much independence needed? Wrong question

Chaining $\mathbf{E}[t] = O(1)$ $\mathbf{E}[t^k] = O(1)$ $\max t = O(\lg n / \lg\lg n)$	2 $2k+1$ $\Theta(\lg n / \lg\lg n)$	
Linear probing	≤ 5 [Pagh[2], Ružić'07]	≥ 5 **[PT ICALP'10]**
Cuckoo hashing	$O(\lg n)$	≥ 6 [Cohen, Kane'05]
F_2 estimation	4 [Alon, Mathias, Szegedy'99]	
ε-minwise indep.	$O(\lg \frac{1}{\varepsilon})$ [Indyk'99]	$\Omega(\lg \frac{1}{\varepsilon})$ **[PT ICALP'10]**

Patrascu-T'11: Despite its 4-dependence, simple tabulation suffices for all the above applications:

One simple and fast hashing scheme for many needs.

Still many applications missing, e.g., for statistics, if we toss n coins, we want number of tails to be concentrated around $n/2$. Also, above, the O-notation hides large constant factors depending exponentially in c.

Zero sets and linear independence

Key set $Z \neq \emptyset$ is a zero set if in each position every character occurs an even number of times, i.e.,

$$\forall i, a \in [c] \times \Sigma : |\{x \in Z | x_i = a\}| \text{ is even.}$$

E.g., $\{a_1 a_2, a_1 b_2, b_1 a_2, b_1 b_2)\} \subset \Sigma^2$.

Zero sets and linear independence

Key set $Z \neq \emptyset$ is a zero set if in each position every character occurs an even number of times, i.e.,

$$\forall i, a \in [c] \times \Sigma : |\{x \in Z | x_i = a\}| \text{ is even.}$$

E.g., $\{a_1 a_2, a_1 b_2, b_1 a_2, b_1 b_2)\} \subset \Sigma^2$.

If Z is a zero set, then for any simple tabulation function h,

$$\bigoplus_{x \in Z} h(x) = 0.$$

Zero sets and linear independence

Key set $Z \neq \emptyset$ is a zero set if in each position every character occurs an even number of times, i.e.,

$$\forall i, a \in [c] \times \Sigma : |\{x \in Z | x_i = a\}| \text{ is even.}$$

E.g., $\{a_1 a_2, a_1 b_2, b_1 a_2, b_1 b_2)\} \subset \Sigma^2$.

If Z is a zero set, then for any simple tabulation function h,

$$\bigoplus_{x \in Z} h(x) = 0.$$

Key set X linearly indepedent if it does not contain a zero set.

Zero sets and linear independence

Key set $Z \neq \emptyset$ is a zero set if in each position every character occurs an even number of times, i.e.,

$$\forall i, a \in [c] \times \Sigma : |\{x \in Z | x_i = a\}| \text{ is even.}$$

E.g., $\{a_1 a_2, a_1 b_2, b_1 a_2, b_1 b_2)\} \subset \Sigma^2$.

If Z is a zero set, then for any simple tabulation function h,

$$\bigoplus_{x \in Z} h(x) = 0.$$

Key set X linearly indepedent if it does not contain a zero set.

Lemma Key set $X \subseteq \Sigma^c$ is linearly independent if and only if simple tabulation hashing is fully random on X.

Zero sets and linear independence

Key set $Z \neq \emptyset$ is zero set if

$$\forall i, a \in [c] \times \Sigma : |\{x \in Z | x_i = a\}| \text{ is even.}$$

Key set X linearly indepedent if it does not contain a zero set.

Lemma Key set $X \subseteq \Sigma^c$ is linearly independent if and only if simple tabulation hashing is fully random on X.

Zero sets and linear independence

Key set $Z \neq \emptyset$ is zero set if

$$\forall i, a \in [c] \times \Sigma : |\{x \in Z | x_i = a\}| \text{ is even.}$$

Key set X linearly indepedent if it does not contain a zero set.

Lemma Key set $X \subseteq \Sigma^c$ is linearly independent if and only if simple tabulation hashing is fully random on X.

Observation If c and Σ not small, random key set likely to be linearly independent.

Zero sets and linear independence

Key set $Z \neq \emptyset$ is zero set if

$$\forall i, a \in [c] \times \Sigma : |\{x \in Z | x_i = a\}| \text{ is even.}$$

Key set X linearly indepedent if it does not contain a zero set.

Lemma Key set $X \subseteq \Sigma^c$ is linearly independent if and only if simple tabulation hashing is fully random on X.

Observation If c and Σ not small, random key set likely to be linearly independent.

Idea Use randomizing function $\tilde{h} : \Sigma^c \to \Sigma^{c+d}$ to create derived keys $\tilde{x} = \tilde{h}(x)$, so for given key set X, w.h.p.,

$$\tilde{X} = \{\tilde{h}(x) | x \in X\} \text{ is linearly independent.}$$

Then for simple tabulation $\hat{h} : \Sigma^{c+d} \to [2]^b$,

$$\hat{h} \circ \tilde{h} \text{ is fully-random on } X.$$

Tornado hashing

Idea Use randomizing function $\tilde{h} : \Sigma^c \rightarrow \Sigma^{c+d}$ to create derived keys $\tilde{x} = \tilde{h}(x)$, so for given key set X, w.h.p.,

$$\tilde{X} = \{\tilde{h}(x) | x \in X\} \text{ is linearly independent.}$$

Then for simple tabulation $\hat{h} : \Sigma^{c+d} \rightarrow [2]^b$,

$$\hat{h} \circ \tilde{h} \text{ is fully-random on } X.$$

Tornado hashing

Idea Use randomizing function $\tilde{h} : \Sigma^c \to \Sigma^{c+d}$ to create derived keys $\tilde{x} = \tilde{h}(x)$, so for given key set X, w.h.p.,

$$\tilde{X} = \{\tilde{h}(x) | x \in X\} \text{ is linearly independent.}$$

Then for simple tabulation $\hat{h} : \Sigma^{c+d} \to [2]^b$,

$$\hat{h} \circ \tilde{h} \text{ is fully-random on } X.$$

We introduce tornado tabulation hashing defining randomizer $\tilde{h}$ such that $h = \hat{h} \circ \tilde{h}$ can be computed by $c + d$ lookups in tables of size Σ and such that if $|\Sigma| \geq \max\{2|X|, 256\}$, then with probability at least

$$1 - 7|X|^3(3/|\Sigma|)^{d+1} - 1/2^{|\Sigma|/2} .$$

$\tilde{X}$ is linearly independent implying h is fully-random on X

Tornado hashing

Idea Use randomizing function $\tilde{h} : \Sigma^c \to \Sigma^{c+d}$ to create derived keys $\tilde{x} = \tilde{h}(x)$, so for given key set X, w.h.p.,

$$\tilde{X} = \{\tilde{h}(x) | x \in X\} \text{ is linearly independent.}$$

Then for simple tabulation $\hat{h} : \Sigma^{c+d} \to [2]^b$,

$$\hat{h} \circ \tilde{h} \text{ is fully-random on } X.$$

We introduce tornado tabulation hashing defining randomizer $\tilde{h}$ such that $h = \hat{h} \circ \tilde{h}$ can be computed by $c + d$ lookups in tables of size Σ and such that if $|\Sigma| \geq \max\{2|X|, 256\}$, then with probability at least

$$1 - 7|X|^3(3/|\Sigma|)^{d+1} - 1/2^{|\Sigma|/2} .$$

$\tilde{X}$ is linearly independent implying h is fully-random on X.

Previous best Mixed Tabulation Hashing had error probabilty

$$\left(O(cd)^c/|\Sigma|\right)^{\lfloor d/2 \rfloor - 1} + 1/2^{\Omega(|\Sigma|)}.$$

Tornado hashing

Idea Use randomizing function $\tilde{h} : \Sigma^c \to \Sigma^{c+d}$ to create derived keys $\tilde{x} = \tilde{h}(x)$, so for given key set X, w.h.p.,

$$\tilde{X} = \{\tilde{h}(x) | x \in X\} \text{ is linearly independent.}$$

Then for simple tabulation $\hat{h} : \Sigma^{c+d} \to [2]^b$,

$$\hat{h} \circ \tilde{h} \text{ is fully-random on } X.$$

We introduce tornado tabulation hashing defining randomizer $\tilde{h}$ such that $h = \hat{h} \circ \tilde{h}$ can be computed by $c + d$ lookups in tables of size Σ and such that if $|\Sigma| \geq \max\{2|X|, 256\}$, then with probability at least

$$1 - 7|X|^3(3/|\Sigma|)^{d+1} - 1/2^{|\Sigma|/2} \; .$$

$\tilde{X}$ is linearly independent implying h is fully-random on X.

Previous best Mixed Tabulation Hashing had error probabilty

$$(\underbrace{O(cd)^c/|\Sigma|}_{\text{above } 1?})^{\lfloor d/2 \rfloor - 1} + 1/2^{\Omega(|\Sigma|)}.$$

C-code for Tornado Tabulation Hashing

```
#include <stdint.h>
uint32_t Tornado(uint32_t x, uint64_t[8][256] H) {
  uint32_t i; uint64_t h=0; uint8_t a;
  for (i=0;i<3;i++) {
    a=x;
    x>>=8;
    h^=H[i][a];}
  h^=x;
  for (i=3;i<8;i++) {
    a=h;
    h>>=8;
    h^=H[i][a];}
  h=&((1<<24)-1);
  return ((uint32_t) h);}
```

Local Uniformity

- Tornado Hashing has a local uniformity that in many cases imply that we can relax the condition $|\Sigma| \geq 2|X|$ needed for fully random hashing on X.

Local Uniformity

- Tornado Hashing has a local uniformity that in many cases imply that we can relax the condition $|\Sigma| \geq 2|X|$ needed for fully random hashing on X.
- For linear probing, to get within a factor $(1 + o(1))$ Knuth's bound $(1 + 1/\varepsilon^2)/2$ on the expected search length, it suffices with $|\Sigma| \geq ((\lg |X|)/\varepsilon)^2$.

Local Uniformity

- Tornado Hashing has a local uniformity that in many cases imply that we can relax the condition $|\Sigma| \geq 2|X|$ needed for fully random hashing on X.
- For linear probing, to get within a factor $(1 + o(1))$ Knuth's bound $(1 + 1/\varepsilon^2)/2$ on the expected search length, it suffices with $|\Sigma| \geq ((\lg |X|)/\varepsilon)^2$.
- For vector k-sample based on one-permutation, w.h.p

$$1 - 7|X|^3(3/|\Sigma|)^{d+1} - 1/2^{|\Sigma|/2} .$$

we get almost as good as fully random if $|\Sigma| \geq \max\{7k \lg(dk|\Sigma|), d2^{16}\}$.

Local Uniformity: Under the Hood

Technical Theorem

- Let $h : \Sigma^c \to \{0,1\}^\ell$ be a tornado tabulation hash function with d derived characters.
- Let $D \subseteq [\ell]$ be set of output bit positions, called select bits.
- Let $h^{|D}$ be h outputting only hash bits from positions in D.
- Let M be a vector of $|D|$ bits
- For a given key set $X \subseteq \Sigma^c$, h selects a subset $S = \{x \in X | s(h^{|D}(x)) = M\}$.
- If $|\Sigma| \geq \max\{2^8, \mathbf{E}[|S|]\}$, then with probability

$$1 - 7|X|^3(3/|\Sigma|)^{d+1} - 1/2^{|\Sigma|/2} \ (= \text{w.h.p}),$$

all non-select hash bits from S are fully random.

Local Uniformity: Under the Hood

Technical Theorem

- Let $h : \Sigma^c \to \{0,1\}^\ell$ be a tornado tabulation hash function with d derived characters.
- Let $D \subseteq [\ell]$ be set of output bit positions, called select bits.
- Let $h^{|D}$ be h outputting only hash bits from positions in D.
- Let M be a vector of $|D|$ bits
- For a given key set $X \subseteq \Sigma^c$, h selects a subset $S = \{x \in X | s(h^{|D}(x)) = M\}$.
- If $|\Sigma| \geq \max\{2^8, \mathbf{E}[|S|]\}$, then with probability

$$1 - 7|X|^3(3/|\Sigma|)^{d+1} - 1/2^{|\Sigma|/2} \ (= \text{w.h.p}),$$

all non-select hash bits from S are fully random.

Note: If $D = \emptyset$ then $S = X$ and then h fully random on X.

- If moreover $|\Sigma| > d\,2^{16}$ and $\mu = \mathbf{E}[|S|]$, then, w.h.p.,

$$\Pr[||S| - \mu|] \geq \delta\mu] \leq 4\exp(-\mu\delta^2/7)$$

Fast Vector *k*-Sampling with Tornado Tabulation

- Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.

Fast Vector k-Sampling with Tornado Tabulation

- Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.
- Sampling from disjoint set R and B.

Fast Vector k-Sampling with Tornado Tabulation

- Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.
- Sampling from disjoint set R and B.
- If $|R \cup B| \leq |\Sigma|/2$, then, w.h.p, tornado tabulation fully random, so fully-random fast vector sampling.

Fast Vector k-Sampling with Tornado Tabulation

- Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.
- Sampling from disjoint set R and B.
- If $|R \cup B| \leq |\Sigma|/2$, then, w.h.p, tornado tabulation fully random, so fully-random fast vector sampling.
- If $|R \cup B| > |\Sigma|/2 = 7k \ln \sigma$, then, w.h.p, fast vector k-sampling uses single one-permutation sampling:

One-Permutation All k samples found, so we can stop	96 99	52 5	63	73 27
One-Permutation	63	52 27	99 5	96 73
Another $k-2$ permutations	⋮	⋮	⋮	⋮
	5 99	73 27	52 99	73 5

Fast Vector k-Sampling with Tornado Tabulation

- Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.
- Sampling from disjoint set R and B.
- If $|R \cup B| \leq |\Sigma|/2$, then, w.h.p, tornado tabulation fully random, so fully-random fast vector sampling.
- If $|R \cup B| > |\Sigma|/2 = 7k \ln \sigma$, then, w.h.p, fast vector k-sampling uses single one-permutation sampling.
- Henceforth focus on this one-permutation case.

One-Permutation k-Sample using Tornado Tabulation

Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.

$$H(x) = \overbrace{0010101110010101001}^{b(x) = \text{first } \log k \text{ bits}}\overbrace{00011001010001110010011100}^{h(x) = \text{remaining } \ell - \log k \text{ bits}}$$

$$|R \cup B| > |\Sigma|/2, \; S(R \cup B)_i = \operatorname{argmin}_{x \in R \cup B, b(x) = i} h(x).$$

One-Permutation k-Sample using Tornado Tabulation

Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.

$$H(x) = \overbrace{0010101110010101001}^{b(x) = \text{first } \log k \text{ bits}}\overbrace{\underbrace{00000000}_{z \text{ bits}}0100011100100111 00}^{h(x) = \text{remaining } \ell - \log k \text{ bits}}$$

$$|R \cup B| > |\Sigma|/2, \; S(R \cup B)_i = \text{argmin}_{x \in R \cup B, b(x)=i} h(x).$$

- Key x small if $z = \log(|R \cup B|/(7k \ln \sigma))$ leading zeros in $h(x)$, i.e, $h(x) < 1/2^z$.

One-Permutation k-Sample using Tornado Tabulation

Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.

$$H(x) = \overbrace{0010101110010101001}^{b(x)\,=\,\text{first}\,\log k\ \text{bits}}\overbrace{\underbrace{00000000}_{z\ \text{bits}}010001110010011100}^{h(x)\,=\,\text{remaining}\ \ell - \log k\ \text{bits}}$$

$$|R \cup B| > |\Sigma|/2,\ \ S(R \cup B)_i = \text{argmin}_{x \in R \cup B, b(x)=i} h(x).$$

- Key x small if $z = \log(|R \cup B|/(7k \ln \sigma))$ leading zeros in $h(x)$, i.e, $h(x) < 1/2^z$. Prob. $p_z = 2^{-z} \leq (7k \ln \sigma)/|R \cup B|$.

One-Permutation k-Sample using Tornado Tabulation

Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.

$$H(x) = \overbrace{0010101110010101001}^{b(x)\,=\,\text{first }\log k\text{ bits}}\overbrace{\underbrace{00000000}_{z\text{ bits}}01000111001001100}^{h(x)\,=\,\text{remaining }\ell - \log k\text{ bits}}$$

$$|R \cup B| > |\Sigma|/2,\ S(R \cup B)_i = \text{argmin}_{x \in R \cup B, b(x)=i} h(x).$$

- Key x small if $z = \log(|R \cup B|/(7k \ln \sigma))$ leading zeros in $h(x)$, i.e, $h(x) < 1/2^z$. Prob. $p_z = 2^{-z} \leq (7k \ln \sigma)/|R \cup B|$.
- For each bucket i, expected number of small keys $x \in R \cup B$ in i is $\mu_{z,i} = |R \cup B| p_z / k \approx 7 \ln \sigma \ll |\Sigma|/2$.

One-Permutation k-Sample using Tornado Tabulation

Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.

$$H(x) = \overbrace{001010111001010100 1}^{b(x)\,=\,\text{first}\ \log k\ \text{bits}}\overbrace{\underbrace{00000000}_{z\ \text{bits}}0100011100100 11100}^{h(x)\,=\,\text{remaining}\ \ell-\log k\ \text{bits}}$$

$$|R \cup B| > |\Sigma|/2, \;\; S(R \cup B)_i = \text{argmin}_{x \in R \cup B, b(x)=i} h(x).$$

- Key x small if $z = \log(|R \cup B|/(7k \ln \sigma))$ leading zeros in $h(x)$, i.e, $h(x) < 1/2^z$. Prob. $p_z = 2^{-z} \leq (7k \ln \sigma)/|R \cup B|$.
- For each bucket i, expected number of small keys $x \in R \cup B$ in i is $\mu_{z,i} = |R \cup B| p_z/k \approx 7 \ln \sigma \ll |\Sigma|/2$.
- By Tornado Chernoff, bucket i has small key with probability $1 - \exp(-\mu_{z,i}/7) = 1/\sigma$.

One-Permutation k-Sample using Tornado Tabulation

Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.

$$H(x) = \overbrace{001010111001010100 1}^{b(x) = \text{first } \log k \text{ bits}}\overbrace{\underbrace{00000000}_{z \text{ bits}}010001110010011100}^{h(x) = \text{remaining } \ell - \log k \text{ bits}}$$

$$|R \cup B| > |\Sigma|/2, \; S(R \cup B)_i = \text{argmin}_{x \in R \cup B, b(x)=i} h(x).$$

- Key x small if $z = \log(|R \cup B|/(7k \ln \sigma))$ leading zeros in $h(x)$, i.e, $h(x) < 1/2^z$. Prob. $p_z = 2^{-z} \leq (7k \ln \sigma)/|R \cup B|$.
- For each bucket i, expected number of small keys $x \in R \cup B$ in i is $\mu_{z,i} = |R \cup B| p_z / k \approx 7 \ln \sigma \ll |\Sigma|/2$.
- By Tornado Chernoff, bucket i has small key with probability $1 - \exp(-\mu_{z,i}/7) = 1/\sigma$.
- With probability $1 - k/\sigma > 1/|\Sigma|^{d-1}$, every bucket has a small key, so all samples $S(R \cup B)_i$ are small.

One-Permutation k-Sample using Tornado Tabulation

Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.

$$H(x) = \overbrace{001010111001010100 1}^{b(x) = \text{first } \log k \text{ bits}}\overbrace{\underbrace{00000000}_{z \text{ bits}}010001110010011100}^{h(x) = \text{remaining } \ell - \log k \text{ bits}}$$

$|R \cup B| > |\Sigma|/2, \; S(R \cup B)_i = \text{argmin}_{x \in R \cup B, b(x)=i} h(x).$

- Key x small if $z = \log_2(|R \cup B|/(7k \ln \sigma))$ leading zeros in $h(x)$. Prob $p_z = 2^{-z} \leq (7k \ln \sigma)/|R \cup B|$.
- W.h.p, all samples $S(R \cup B)_i$ are small.

One-Permutation k-Sample using Tornado Tabulation

Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.

$$H(x) = \overbrace{0010101110010101001}^{b(x) = \text{first } \log k \text{ bits}}\overbrace{\underbrace{00000000}_{z \text{ bits}}0100011100100111 00}^{h(x) = \text{remaining } \ell - \log k \text{ bits}}$$

$$|R \cup B| > |\Sigma|/2, \;\; S(R \cup B)_i = \text{argmin}_{x \in R \cup B, b(x) = i} h(x).$$

- Key x small if $z = \log_2(|R \cup B|/(7k \ln \sigma))$ leading zeros in $h(x)$. Prob $p_z = 2^{-z} \leq (7k \ln \sigma)/|R \cup B|$.
- W.h.p, all samples $S(R \cup B)_i$ are small.
- Assume $|R| = |B|$ (will be removed later).

One-Permutation k-Sample using Tornado Tabulation

Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.

$$H(x) = \overbrace{0010101110010101001}^{b(x) = \text{first } \log k \text{ bits}}\overbrace{\underbrace{00000000}_{z \text{ bits}}\,0100011100100111 00}^{h(x) = \text{remaining } \ell - \log k \text{ bits}}$$

$$|R \cup B| > |\Sigma|/2, \;\; S(R \cup B)_i = \operatorname{argmin}_{x \in R \cup B, b(x) = i} h(x).$$

- Key x small if $z = \log_2(|R \cup B|/(7k \ln \sigma))$ leading zeros in $h(x)$. Prob $p_z = 2^{-z} \leq (7k \ln \sigma)/|R \cup B|$.
- W.h.p, all samples $S(R \cup B)_i$ are small.
- Assume $|R| = |B|$ (will be removed later).
- R_z = set of small red balls.

One-Permutation k-Sample using Tornado Tabulation

Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.

$$H(x) = \overbrace{0010101110010101001}^{b(x)\,=\,\text{first}\log k\text{ bits}}\overbrace{\underbrace{00000000}_{z\text{ bits}}0100011100100111100}^{h(x)\,=\,\text{remaining }\ell-\log k\text{ bits}}$$

$$|R \cup B| > |\Sigma|/2,\ S(R \cup B)_i = \text{argmin}_{x \in R \cup B, b(x)=i} h(x).$$

- Key x small if $z = \log_2(|R \cup B|/(7k \ln \sigma))$ leading zeros in $h(x)$. Prob $p_z = 2^{-z} \leq (7k \ln \sigma)/|R \cup B|$.
- W.h.p, all samples $S(R \cup B)_i$ are small.
- Assume $|R| = |B|$ (will be removed later).
- R_z = set of small red balls.
- Then $\mathbf{E}[|R_z|] = p_z|R| = 3.5k(\ln \sigma) \ll |\Sigma|/2$, so by Tornado Chernoff, w.h.p,

$$|R_z| = \left(1 \pm \sqrt{\frac{7 \ln \sigma}{\mathbf{E}[|R_z|]}}\right)\mathbf{E}[|R_z|] = (1 \pm \sqrt{2/k})p_z|R|.$$

One-Permutation k-Sample using Tornado Tabulation

Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.

$$H(x) = \overbrace{0010101110010101001}^{b(x)\,=\,\text{first } \log k \text{ bits}}\overbrace{\underbrace{00000000}_{z \text{ bits}}0100011100100111 00}^{h(x)\,=\,\text{remaining } \ell - \log k \text{ bits}}$$

$$|R \cup B| > |\Sigma|/2, \; S(R \cup B)_i = \text{argmin}_{x \in R \cup B, b(x)=i} h(x).$$

- Key x small if $z = \log_2(|R \cup B|/(7k \ln \sigma))$ leading zeros in $h(x)$. Prob $p_z = 2^{-z} \leq (7k \ln \sigma)/|R \cup B|$.
- Assume $|R| = |B|$ (will be removed later).
- R_z / B_z = small red / blue balls.

One-Permutation k-Sample using Tornado Tabulation

Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.

$$H(x) = \overbrace{0010101110010101001}^{b(x) = \text{first } \log k \text{ bits}}\overbrace{\underbrace{00000000}_{z \text{ bits}}01000111001001100}^{h(x) = \text{remaining } \ell - \log k \text{ bits}}$$

$$|R \cup B| > |\Sigma|/2, \;\; S(R \cup B)_i = \text{argmin}_{x \in R \cup B, b(x) = i} h(x).$$

- Key x small if $z = \log_2(|R \cup B|/(7k \ln \sigma))$ leading zeros in $h(x)$. Prob $p_z = 2^{-z} \leq (7k \ln \sigma)/|R \cup B|$.
- Assume $|R| = |B|$ (will be removed later).
- R_z / B_z = small red / blue balls.
- By Tornado Chernoff, w.h.p, $|R_z| = (1 \pm \sqrt{2/k}) p_z |R|$

One-Permutation k-Sample using Tornado Tabulation

Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.

$$H(x) = \overbrace{0010101110010101001}^{b(x)\,=\,\text{first } \log k \text{ bits}}\overbrace{\underbrace{00000000}_{z \text{ bits}}0100011100100111000}^{h(x)\,=\,\text{remaining } \ell - \log k \text{ bits}}$$

$$|R \cup B| > |\Sigma|/2, \;\; S(R \cup B)_i = \text{argmin}_{x \in R \cup B, b(x)=i} h(x).$$

- Key x small if $z = \log_2(|R \cup B|/(7k \ln \sigma))$ leading zeros in $h(x)$. Prob $p_z = 2^{-z} \leq (7k \ln \sigma)/|R \cup B|$.
- Assume $|R| = |B|$ (will be removed later).
- R_z / B_z = small red / blue balls.
- By Tornado Chernoff, w.h.p, $|R_z| = (1 \pm \sqrt{2/k})p_z|R|$ and $|R_z \cup B_z| = (1 \pm \sqrt{1/k})p_z|R \cup B|$,

One-Permutation k-Sample using Tornado Tabulation

Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.

$$H(x) = \overbrace{0010101110010101001}^{b(x) = \text{first } \log k \text{ bits}}\overbrace{\underbrace{00000000}_{z \text{ bits}}0100011100100 11100}^{h(x) = \text{remaining } \ell - \log k \text{ bits}}$$

$$|R \cup B| > |\Sigma|/2, \; S(R \cup B)_i = \text{argmin}_{x \in R \cup B, b(x) = i} h(x).$$

- Key x small if $z = \log_2(|R \cup B|/(7k \ln \sigma))$ leading zeros in $h(x)$. Prob $p_z = 2^{-z} \leq (7k \ln \sigma)/|R \cup B|$.
- Assume $|R| = |B|$ (will be removed later).
- R_z / B_z = small red / blue balls.
- By Tornado Chernoff, w.h.p, $|R_z| = (1 \pm \sqrt{2/k})p_z|R|$ and $|R_z \cup B_z| = (1 \pm \sqrt{1/k})p_z|R \cup B|$, so $f_z = |R_z|/|R_z \cup B_z| = (1 \pm 3/\sqrt{1/k})f$.

One-Permutation k-Sample using Tornado Tabulation

Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.

$$H(x) = \overbrace{0010101110010101001}^{b(x) = \text{first } \log k \text{ bits}}\overbrace{\underbrace{00000000}_{z \text{ bits}}0100011100100111100}^{h(x) = \text{remaining } \ell - \log k \text{ bits}}$$

$|R \cup B| > |\Sigma|/2, \; S(R \cup B)_i = \text{argmin}_{x \in R \cup B, b(x)=i} h(x).$

- Key x small if $z = \log_2(|R \cup B|/(7k \ln \sigma))$ leading zeros in $h(x)$. W.h.p, all samples $S(R \cup B)_i$ are small.
- Assume $|R| = |B|$ (will be removed later).
- R_z / B_z = small red / blue balls.
- W.h.p, $f_z = |R_z|/|R_z \cup B_z| = (1 \pm 3/\sqrt{k})f$.

One-Permutation k-Sample using Tornado Tabulation

Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.

$$H(x) = \overbrace{0010101110010101001}^{b(x)\,=\,\text{first }\log k\text{ bits}}\overbrace{\underbrace{00000000}_{z\text{ bits}}0100011100100 11100}^{h(x)\,=\,\text{remaining }\ell-\log k\text{ bits}}$$

$$|R \cup B| > |\Sigma|/2, \;\; S(R \cup B)_i = \text{argmin}_{x \in R \cup B, b(x)=i} h(x).$$

- Key x small if $z = \log_2(|R \cup B|/(7k \ln \sigma))$ leading zeros in $h(x)$. W.h.p, all samples $S(R \cup B)_i$ are small.
- Assume $|R| = |B|$ (will be removed later).
- R_z / B_z = small red / blue balls.
- W.h.p, $f_z = |R_z|/|R_z \cup B_z| = (1 \pm 3/\sqrt{k})f$.
- $\mathbf{E}[|R_z \cup B_z|] \leq |\Sigma|/2$, so by Tornado Local Uniformity, w.h.p, all other output bits of H are fully random on $R_z \cup B_z$.

One-Permutation k-Sample using Tornado Tabulation

Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.

$$H(x) = \overbrace{0010101110010101001}^{b(x) = \text{first } \log k \text{ bits}}\overbrace{\underbrace{00000000}_{z \text{ bits}}010001110010011100}^{h(x) = \text{remaining } \ell - \log k \text{ bits}}$$

$|R \cup B| > |\Sigma|/2, \; S(R \cup B)_i = \text{argmin}_{x \in R \cup B, b(x)=i} h(x).$

- Key x small if $z = \log_2(|R \cup B|/(7k \ln \sigma))$ leading zeros in $h(x)$. W.h.p, all samples $S(R \cup B)_i$ are small.
- Assume $|R| = |B|$ (will be removed later).
- R_z / B_z = small red / blue balls.
- W.h.p, $f_z = |R_z|/|R_z \cup B_z| = (1 \pm 3/\sqrt{k})f$.
- $\mathbf{E}[|R_z \cup B_z|] \leq |\Sigma|/2$, so by Tornado Local Uniformity, w.h.p, all other output bits of H are fully random on $R_z \cup B_z$.
- Then S fully random k-sample without replacement from $R_z \cup B_z$,

One-Permutation k-Sample using Tornado Tabulation

Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.

$$H(x) = \overbrace{0010101110010101001}^{b(x)=\text{first }\log k\text{ bits}}\overbrace{\underbrace{00000000}_{z\text{ bits}}0100011100100011100}^{h(x)=\text{remaining }\ell-\log k\text{ bits}}$$

$$|R \cup B| > |\Sigma|/2,\ \ S(R \cup B)_i = \text{argmin}_{x \in R \cup B, b(x)=i} h(x).$$

- Key x small if $z = \log_2(|R \cup B|/(7k \ln \sigma))$ leading zeros in $h(x)$. W.h.p, all samples $S(R \cup B)_i$ are small.
- Assume $|R| = |B|$ (will be removed later).
- R_z / B_z = small red / blue balls.
- W.h.p, $f_z = |R_z|/|R_z \cup B_z| = (1 \pm 3/\sqrt{k})f$.
- $\mathbf{E}[|R_z \cup B_z|] \leq |\Sigma|/2$, so by Tornado Local Uniformity, w.h.p, all other output bits of H are fully random on $R_z \cup B_z$.
- Then S fully random k-sample without replacement from $R_z \cup B_z$, so by Fully Random Chernoff, w.h.p, $|S \cap R_z| = (1 \pm \sqrt{3 \ln \sigma/(f_z k)})\, k f_z$.

One-Permutation k-Sample using Tornado Tabulation

Tornado table size $|\Sigma| \geq 14k \ln \sigma$ where $\sigma = |\Sigma|^d$.

$$H(x) = \overbrace{001010111001010100 1}^{b(x) = \text{first } \log k \text{ bits}}\overbrace{\underbrace{00000000}_{z \text{ bits}}010001110010011100}^{h(x) = \text{remaining } \ell - \log k \text{ bits}}$$

$$|R \cup B| > |\Sigma|/2, \; S(R \cup B)_i = \text{argmin}_{x \in R \cup B, b(x)=i} h(x).$$

- Key x small if $z = \log_2(|R \cup B|/(7k \ln \sigma))$ leading zeros in $h(x)$. W.h.p, all samples $S(R \cup B)_i$ are small.
- Assume $|R| = |B|$ (will be removed later).
- R_z / B_z = small red / blue balls.
- W.h.p, $f_z = |R_z|/|R_z \cup B_z| = (1 \pm 3/\sqrt{k})f$.
- $\mathbf{E}[|R_z \cup B_z|] \leq |\Sigma|/2$, so by Tornado Local Uniformity, w.h.p, all other output bits of H are fully random on $R_z \cup B_z$.
- Then S fully random k-sample without replacement from $R_z \cup B_z$, so by Fully Random Chernoff, w.h.p, $|S \cap R_z| = (1 \pm \sqrt{3 \ln \sigma/(f_z k)})\, k f_z$.
- $|S \cap R| = |S \cap R_z| = (1 \pm \sqrt{(3 \ln \sigma + 9)/(fk)})\, kf$.

$$H(x) = \overbrace{0010101110010101001}^{b(x)\,=\,\text{first }\log k\text{ bits}}\overbrace{\underbrace{\underbrace{00000}_{\text{x bits}}001}_{z\text{ bits}}\,0100011100100011100}^{h(x)\,=\,\text{remaining }\ell-\log k\text{ bits}}$$

Assume $|R| \leq |B|$

$x = \lceil \log_2(|R|/(7k \ln \sigma)\rceil$, $p_x = 2^{-x}$, $p_x|R| = 7k \log k$.

$z = \lceil \log_2(|R \cup B|/(7k \ln \sigma)\rceil$, $p_z = 2^{-z}$, $p_z|R \cup B| = 7k \log k$.

$$H(x) = \overbrace{001010111001010101001}^{b(x) = \text{first } \log k \text{ bits}}\overbrace{\underbrace{\underbrace{00000}_{x \text{ bits}}001}_{z \text{ bits}}\,010001110010011100}^{h(x) = \text{remaining } \ell - \log k \text{ bits}}$$

Assume $|R| \leq |B|$

$x = \lceil \log_2(|R|/(7k \ln \sigma) \rceil$, $p_x = 2^{-x}$, $p_x|R| = 7k \log k$.

$z = \lceil \log_2(|R \cup B|/(7k \ln \sigma) \rceil$, $p_z = 2^{-z}$, $p_z|R \cup B| = 7k \log k$.

W.h.p.

- All k samples from S have z leading zeros in local hash.

$$H(x) = \overbrace{0010101110010101001}^{b(x) = \text{first } \log k \text{ bits}}\overbrace{\underbrace{\underbrace{00000}_{\text{x bits}}001}_{z \text{ bits}}\ 010001110010011100}^{h(x) = \text{remaining } \ell - \log k \text{ bits}}$$

Assume $|R| \leq |B|$
$x = \lceil \log_2(|R|/(7k \ln \sigma)) \rceil,\ p_x = 2^{-x},\ p_x|R| = 7k \log k.$
$z = \lceil \log_2(|R \cup B|/(7k \ln \sigma)) \rceil,\ p_z = 2^{-z},\ p_z|R \cup B| = 7k \log k.$
W.h.p.

- All k samples from S have z leading zeros in local hash.
- R_x / B_z = red / blue balls with x / z leading zeros.
- $|R_x| = p_x|R|(1 \pm \sqrt{1/k})$ and $|B_z| = p_z|B|(1 \pm \sqrt{1/k})$

$$H(x) = \overbrace{001010111001010100 1}^{b(x)\,=\,\text{first }\log k\text{ bits}}\overbrace{\underbrace{\underbrace{00000}_{x\text{ bits}}001}_{z\text{ bits}}\;0100011100100 11100}^{h(x)\,=\,\text{remaining }\ell-\log k\text{ bits}}$$

Assume $|R| \le |B|$

$x = \lceil \log_2(|R|/(7k \ln \sigma) \rceil$, $p_x = 2^{-x}$, $p_x|R| = 7k \log k$.

$z = \lceil \log_2(|R \cup B|/(7k \ln \sigma) \rceil$, $p_z = 2^{-z}$, $p_z|R \cup B| = 7k \log k$.

W.h.p.

- All k samples from S have z leading zeros in local hash.
- R_x / B_z = red / blue balls with x / z leading zeros.
- $|R_x| = p_x|R|(1 \pm \sqrt{1/k})$ and $|B_z| = p_z|B|(1 \pm \sqrt{1/k})$
- Local output bits $x+1, \ldots, z$ of H are fully random on R_x.

$$H(x) = \overbrace{0010101110010101001}^{b(x)\,=\,\text{first } \log k \text{ bits}}\overbrace{\underbrace{\underbrace{00000}_{x\text{ bits}}001}_{z\text{ bits}}\,010001110010011100}^{h(x)\,=\,\text{remaining } \ell - \log k \text{ bits}}$$

Assume $|R| \leq |B|$

$x = \lceil \log_2(|R|/(7k \ln \sigma) \rceil$, $p_x = 2^{-x}$, $p_x|R| = 7k \log k$.

$z = \lceil \log_2(|R \cup B|/(7k \ln \sigma) \rceil$, $p_z = 2^{-z}$, $p_z|R \cup B| = 7k \log k$.

W.h.p.

- All k samples from S have z leading zeros in local hash.
- R_x / B_z = red / blue balls with x / z leading zeros.
- $|R_x| = p_x|R|(1 \pm \sqrt{1/k})$ and $|B_z| = p_z|B|(1 \pm \sqrt{1/k})$
- Local output bits $x+1, \ldots, z$ of H are fully random on R_x.
- R_z chosen fully randomly from R_x by zeros in these bits.

$$H(x) = \overbrace{0010101110010101001}^{b(x)\,=\,\text{first } \log k \text{ bits}}\overbrace{\underbrace{\underbrace{00000}_{x \text{ bits}}001}_{z \text{ bits}}\,0100011100100111 00}^{h(x)\,=\,\text{remaining } \ell - \log k \text{ bits}}$$

Assume $|R| \leq |B|$

$x = \lceil \log_2(|R|/(7k \ln \sigma) \rceil$, $p_x = 2^{-x}$, $p_x|R| = 7k \log k$.

$z = \lceil \log_2(|R \cup B|/(7k \ln \sigma) \rceil$, $p_z = 2^{-z}$, $p_z|R \cup B| = 7k \log k$.

W.h.p.

- All k samples from S have z leading zeros in local hash.
- R_x / B_z = red / blue balls with x / z leading zeros.
- $|R_x| = p_x|R|(1 \pm \sqrt{1/k})$ and $|B_z| = p_z|B|(1 \pm \sqrt{1/k})$
- Local output bits $x+1, \ldots, z$ of H are fully random on R_x.
- R_z chosen fully randomly from R_x by zeros in these bits.
- Except first z local output bits, H fully random on $R_z \cup B_z$.

$$H(x) = \overbrace{0010101110010101001}^{b(x)\,=\,\text{first }\log k\text{ bits}}\overbrace{\underbrace{\underbrace{00000}_{x\text{ bits}}001}_{z\text{ bits}}\,0100011100100011100}^{h(x)\,=\,\text{remaining }\ell-\log k\text{ bits}}$$

Assume $|R| \leq |B|$

$x = \lceil \log_2(|R|/(7k \ln \sigma)) \rceil$, $p_x = 2^{-x}$, $p_x|R| = 7k \log k$.

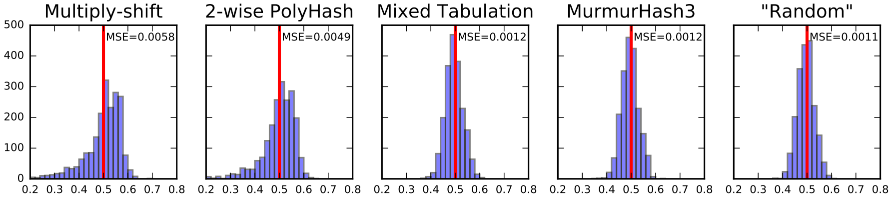
$z = \lceil \log_2(|R \cup B|/(7k \ln \sigma)) \rceil$, $p_z = 2^{-z}$, $p_z|R \cup B| = 7k \log k$.

W.h.p.

- All k samples from S have z leading zeros in local hash.
- R_x / B_z = red / blue balls with x / z leading zeros.
- $|R_x| = p_x|R|(1 \pm \sqrt{1/k})$ and $|B_z| = p_z|B|(1 \pm \sqrt{1/k})$
- Local output bits $x+1, \ldots, z$ of H are fully random on R_x.
- R_z chosen fully randomly from R_x by zeros in these bits.
- Except first z local output bits, H fully random on $R_z \cup B_z$.
- As before, we get
  $|S \cap R| = |S \cap R_z| = (1 \pm \sqrt{(3 \ln \sigma + 9)/k})\, kf$.

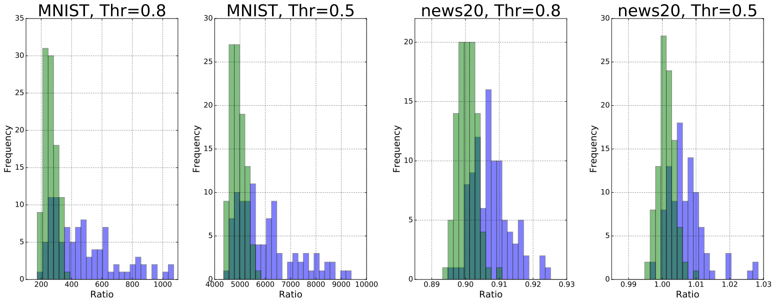
Experiments [Dahlgaard, Knudsen, T, NIPS'17] using Mixed Tabulation, a predecessor of Tornado

Hash function	time ($1..10^7$)	time (News20)
Multiply-shift	7.72 ms	55.78 ms
2-wise PolyHash	17.55 ms	82.47 ms
3-wise PolyHash	42.42 ms	120.19 ms
MurmurHash3	59.70 ms	159.44 ms
CityHash	59.06 ms	162.04 ms
Blake2	3476.31 ms	6408.40 ms
Mixed tabulation	42.98 ms	90.55 ms



Real-World Data inside Locality Sensitive Hashing

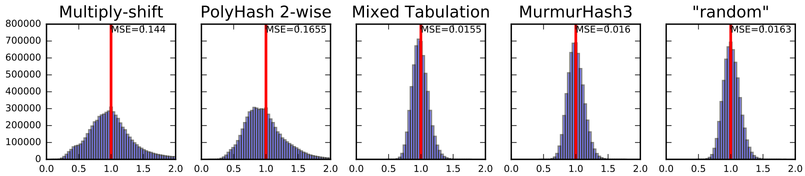
- **MNIST** Standard collection of handwritten digits.
- **News20** Collection of newsgroup documents.



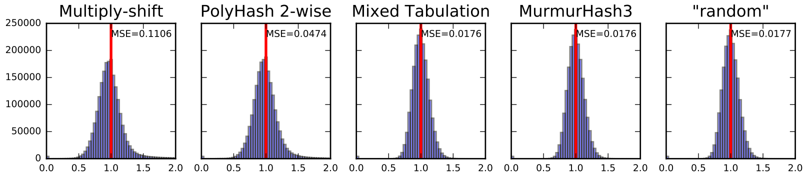
Mixed tabulation smaller (=better) than Multiply-Shift.

Real-World Data inside Feature Hashing

MNIST Standard collection of handwritten digits.



News20 Collection of newsgroup documents.

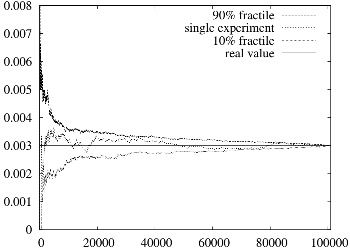


Concluding remarks

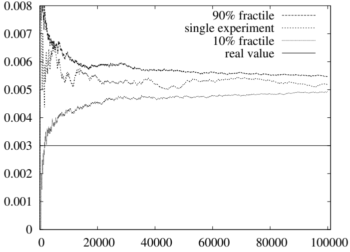
Conclusion on k-mins versus bottom-k

- Both k-mins and bottom-k generalize the idea of storing the smallest hash value.
- With limited independence
  k-mins has major problems with bias whereas bottom-k works perfectly even with 2-independence.

bottom-k



k-mins



- Analysis: union bound over two simple Chebyshev bounds.
- Bottom-k also efficient in steaming: maintain k smallest elements in priority queue in $O(\log k)$ time per element.

Similarity Vector Sample

- Sketch S maps any set A to vector of k samples from A.
- Want S to preserve similarity between sets A and B:
- For each coordinate $i \in [k]$, unbiased estimator

$$\Pr[S(A)_i = S(B)_i] = J(A, B) = |A \cap B| / |A \cup B|.$$

- Let

$$J_S(A, B) = \sum_{i \in [k]} [S(A)_i = S(B)_i] / k.$$

- Want strong concentration of $J_S(A, B)$ around $J(A, B)$.
- The alignment property, that $S(A)_i$ only compared with $S(B)_i$ important, e.g., for Support Vector Machines (SVMs).
- To code similarity as inner product, hash each $S(A)_i$ to b bits with single 1 [Li König CACM'11], e.g, $k = 2$, $b = 4$,

$$\{5, 27, 52, 73, 99\} \rightarrow (72, 5) \rightarrow (0, 0, 0, 1 \textbf{,} 0, 1, 0, 0)$$

- Alignment not satisfied by bottom-k sketch.

Fast vector k-sampling $k = 4$, $A = \{5, 27, 52, 73, 99\}$.

				73
		52		
One-Permutation				27
	99			
		5		
		52		
			99	
One-Permutation			5	
		27		
				73
Another $k-2$ permutations	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	5	73	52	73
	99	27	99	5
k-Mins	27	99	27	52
	52	5	5	99
	73	52	73	27

$S(A) = (99, 52, 99, 73)$.

Fast vector k-sampling $k = 4$, $A = \{5, 27, 52, 73, 99\}$.

One-Permutation	99	52 5		73 27
One-Permutation All k samples found, so we can stop		52 27	99 5	73
Another $k-2$ permutations	⋮	⋮	⋮	⋮
k-Mins	5 99 27 52 73	73 27 99 5 52	52 99 27 5 73	73 5 52 99 27

$S(A) = (99, 52, 99, 73)$.

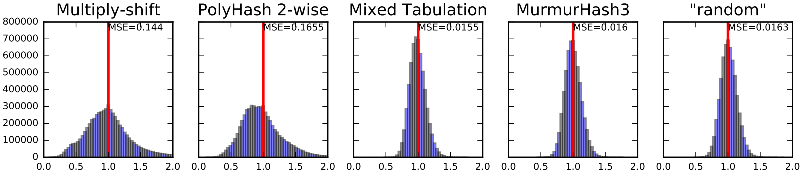
- $O(|A| + k \log k)$ time and Chernoff concentration bounds.

Concentration needs independence between buckets

- Fast Vector Sampling and One-Permutation sampling throws elements into k buckets, and pick a random element from each bucket.
- With full randomness, fraction of samples from R concentrated around $|R|/|R \cup B|$ with Chernoff bounds
- Requires high indpendence between bucket contents.
- Not clear if k independent hashing suffices.
- Generic issue: using hashing to distribute elements in buckets, and aggregating statistics over the buckets to get well-concentrated estimates.
- Idea also used for counting distinct elements [Flajolet et al. FOCS'83], count sketches, and feature hashing.
- Mixed Tabulation Hashing solves the problem quite generically.

Experiments

Hash function	time ($1..10^7$)	time (News20)
Multiply-shift	7.72 ms	55.78 ms
2-wise PolyHash	17.55 ms	82.47 ms
3-wise PolyHash	42.42 ms	120.19 ms
MurmurHash3	59.70 ms	159.44 ms
CityHash	59.06 ms	162.04 ms
Blake2	3476.31 ms	6408.40 ms
Mixed tabulation	42.98 ms	90.55 ms



Open Problem

- Use Tabulation Hashing to solve any problems that in constant time where only solvable with fully-random hashing..

Open Problem

- Use Tabulation Hashing to solve any problems that in constant time where only solvable with fully-random hashing..
- Done for Sparse Johnson-Lindenstrauss. Should be possible for many other types of dimensionality reduction.

Open Problem

- Use Tabulation Hashing to solve any problems that in constant time where only solvable with fully-random hashing..
- Done for Sparse Johnson-Lindenstrauss. Should be possible for many other types of dimensionality reduction.
- The bad thing is that analysis is non-trivial.
- The good thing is that the tabulation hashing is simple and efficient, ready for use in applications.

Open Problem

- Use Tabulation Hashing to solve any problems that in constant time where only solvable with fully-random hashing..
- Done for Sparse Johnson-Lindenstrauss. Should be possible for many other types of dimensionality reduction.
- The bad thing is that analysis is non-trivial.
- The good thing is that the tabulation hashing is simple and efficient, ready for use in applications.
- This contrast the use of too simple hash functions relying on randomness in input.

Thanks for your attention