From previous lecture:

- Universal hashing of complex key universe U, e.g., variable length strings, to integers in $[m]$ with collision probability $2/m$.

From previous lecture:

- Universal hashing of complex key universe U, e.g., variable length strings, to integers in $[m]$ with collision probability $2/m$.
- Essentially one w-bit multiplication per w input bits for $w/2$-bit hash values.

From previous lecture:

- Universal hashing of complex key universe U, e.g., variable length strings, to integers in $[m]$ with collision probability $2/m$.
- Essentially one w-bit multiplication per w input bits for $w/2$-bit hash values.
- For kw-bit hash values, use $2k$ independent hash functions $h_1, ..., h_{2k} : U \to [2^{w/2}]$, and concatenate the outputs:

$$h(x) = h_1(x) \cdots h_{2k}(x).$$

From previous lecture:

- Universal hashing of complex key universe U, e.g., variable length strings, to integers in $[m]$ with collision probability $2/m$.
- Essentially one w-bit multiplication per w input bits for $w/2$-bit hash values.
- For kw-bit hash values, use $2k$ independent hash functions $h_1, ..., h_{2k} : U \to [2^{w/2}]$, and concatenate the outputs:

$$h(x) = h_1(x) \cdots h_{2k}(x).$$

- For few/no collisions over n keys, pick $m \geq n^3$. This is called *universe reduction*.

From previous lecture:

- Universal hashing of complex key universe U, e.g., variable length strings, to integers in $[m]$ with collision probability $2/m$.
- Essentially one w-bit multiplication per w input bits for $w/2$-bit hash values.
- For kw-bit hash values, use $2k$ independent hash functions $h_1, ..., h_{2k} : U \to [2^{w/2}]$, and concatenate the outputs:

$$h(x) = h_1(x) \cdots h_{2k}(x).$$

- For few/no collisions over n keys, pick $m \geq n^3$. This is called *universe reduction*.
- For more complex hash functions H, like k-independent with larger k, we only have to deal with key universe polynomial in number of keys

$$U \xrightarrow{h} [n^3] \xrightarrow{H} R$$

Our focus in this lecture is H.

Fast and Powerful Hashing using Tabulation

Mikkel Thorup
University of Copenhagen

Fast and Powerful Hashing using Tabulation

Mikkel Thorup
University of Copenhagen

Lecture covers results from

- Mihai Pătraşcu adn Mikkel Thorup: The power of simple tabulation hashing. J. ACM 59(3): 14 (2012). Announced at STOC 2011: 1-10
- Mihai Pătraşcu and Mikkel Thorup: Twisted Tabulation Hashing. SODA 2013: 209-228
- Mikkel Thorup: Simple Tabulation, Fast Expanders, Double Tabulation, and High Independence. FOCS 2013: 90-99.
- Søren Dahlgaard and Mikkel Thorup: Approximately Minwise Independence with Twisted Tabulation. SWAT 2014.
- And some recent stuff.

.

Target

- Simple and reliable pseudo-random hashing.

Target

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

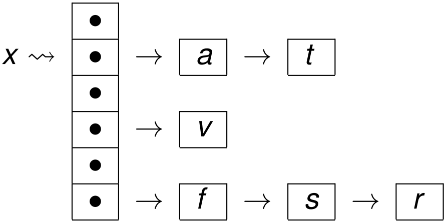
Target

- Simple and reliable pseudo-random hashing.
- Providing algorithmically important probabilisitic guarantees akin to those of truly random hashing, yet easy to implement.
- Bridging theory (assuming truly random hashing) with practice (needing something implementable).

Applications of Hashing $\rightsquigarrow$

Hash tables (n keys and $2n$ hashes: expect 1/2 keys per hash)

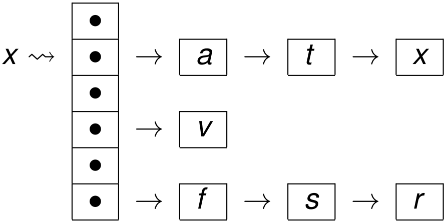
- chaining: follow pointers



Applications of Hashing $\rightsquigarrow$

Hash tables (n keys and $2n$ hashes: expect 1/2 keys per hash)

- chaining: follow pointers



Applications of Hashing $\rightsquigarrow$

Hash tables (n keys and $2n$ hashes: expect 1/2 keys per hash)

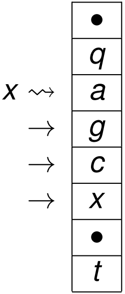
- chaining: follow pointers
- linear probing: sequential search in *one* array

	$\bullet$
	q
$x \rightsquigarrow$	a
$\rightarrow$	g
$\rightarrow$	c
$\rightarrow$	$\bullet$
	$\bullet$
	t

Applications of Hashing $\rightsquigarrow$

Hash tables (n keys and $2n$ hashes: expect 1/2 keys per hash)

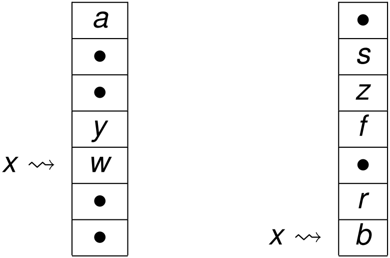
- chaining: follow pointers
- linear probing: sequential search in *one* array



Applications of Hashing ⇝

Hash tables (n keys and $2n$ hashes: expect 1/2 keys per hash)

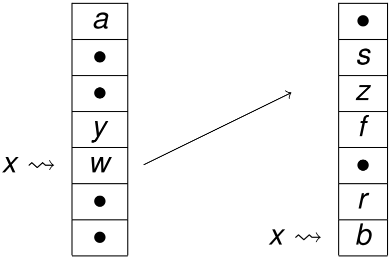
- chaining: follow pointers
- linear probing: sequential search in *one* array
- cuckoo hashing: search $\leq$ 2 locations, complex updates



Applications of Hashing $\rightsquigarrow$

Hash tables (n keys and $2n$ hashes: expect 1/2 keys per hash)

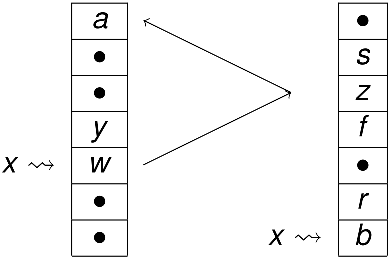
- chaining: follow pointers
- linear probing: sequential search in *one* array
- cuckoo hashing: search ≤ 2 locations, complex updates



Applications of Hashing $\rightsquigarrow$

Hash tables (n keys and $2n$ hashes: expect 1/2 keys per hash)

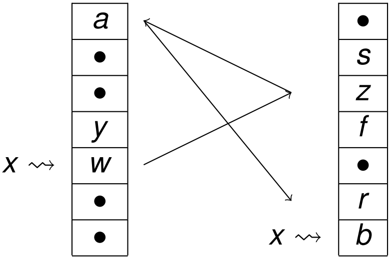
- chaining: follow pointers
- linear probing: sequential search in *one* array
- cuckoo hashing: search ≤ 2 locations, complex updates



Applications of Hashing $\rightsquigarrow$

Hash tables (n keys and $2n$ hashes: expect 1/2 keys per hash)

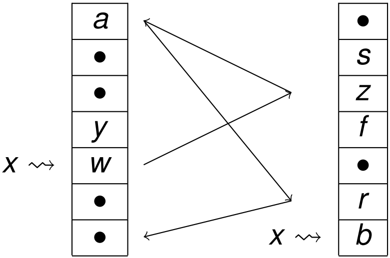
- chaining: follow pointers
- linear probing: sequential search in *one* array
- cuckoo hashing: search ≤ 2 locations, complex updates



Applications of Hashing $\rightsquigarrow$

Hash tables (n keys and $2n$ hashes: expect 1/2 keys per hash)

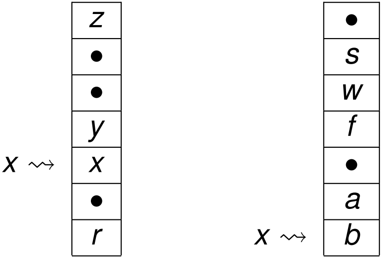
- chaining: follow pointers
- linear probing: sequential search in *one* array
- cuckoo hashing: search $\leq$ 2 locations, complex updates



Applications of Hashing ⇝

Hash tables (n keys and $2n$ hashes: expect 1/2 keys per hash)

- chaining: follow pointers
- linear probing: sequential search in *one* array
- cuckoo hashing: search ≤ 2 locations, complex updates



Applications of Hashing $\rightsquigarrow$

Hash tables (n keys and $2n$ hashes: expect 1/2 keys per hash)

- chaining: follow pointers.
- linear probing: sequential search in *one* array
- cuckoo hashing: search ≤ 2 locations, complex updates

Applications of Hashing $\rightsquigarrow$

Hash tables (n keys and $2n$ hashes: expect 1/2 keys per hash)

- chaining: follow pointers.
- linear probing: sequential search in *one* array
- cuckoo hashing: search ≤ 2 locations, complex updates

Sketching, streaming, and sampling:

- second moment estimation: $F_2(\bar{x}) = \sum_i x_i^2$

Applications of Hashing $\rightsquigarrow$

Hash tables (n keys and $2n$ hashes: expect 1/2 keys per hash)

- chaining: follow pointers.
- linear probing: sequential search in *one* array
- cuckoo hashing: search ≤ 2 locations, complex updates

Sketching, streaming, and sampling:

- second moment estimation: $F_2(\bar{x}) = \sum_i x_i^2$
- sketch A and B to later find $|A \cap B|/|A \cup B|$

$$|A \cap B|/|A \cup B| = \Pr_h[\min h(A) = \min h(B)]$$

We need h to be ε-minwise independent:

$$x \in S: \qquad \Pr[h(x) = \min h(S)] = \frac{1 \pm \varepsilon}{|S|}$$

Applications of Hashing $\rightsquigarrow$

Hash tables (n keys and $2n$ hashes: expect 1/2 keys per hash)

- chaining: follow pointers.
- linear probing: sequential search in *one* array

Important outside theory. These simple practical hash tables often bottlenecks in the processing of data—substantial fraction of worlds computational resources spent here.

Wegman & Carter [FOCS'77]

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

Family $\mathcal{H} = \{h : [u] \to [b]\}$ k-independent iff for random $h \in \mathcal{H}$:

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

Wegman & Carter [FOCS'77]

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

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

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

Classic example: degree $k-1$ polynomial over $\mathbb{Z}_p$, prime p.

- choose $a_0, a_1, \dots, a_{k-1}$ randomly in $[p]$;
- $h(x) = \left(a_0 + a_1 x + \cdots + a_{k-1} x^{k-1}\right) \bmod p$.

Wegman & Carter [FOCS'77]

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

Family $\mathcal{H} = \{h : [u] \to [b]\}$ k-independent iff for random $h \in \mathcal{H}$:

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

Classic example: degree $k-1$ polynomial over $\mathbb{Z}_p$, prime p.

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

Many solutions for k-independent hashing proposed, but generally slow for $k > 3$ and too slow for $k > 5$.

How much independence needed?

Chaining $\mathbf{E}[t] = O(1)$ $\mathbf{E}[t^k] = O(1)$ $t = O\left(\frac{\lg n}{\lg \lg n}\right)$ w.h.p.	2 $2k+1$ $\Theta\left(\frac{\lg n}{\lg \lg n}\right)$	
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.	$O\left(\lg \frac{1}{\varepsilon}\right)$ [Indyk'99]	$\Omega\left(\lg \frac{1}{\varepsilon}\right)$ **[PT ICALP'10]**

How much independence needed?

Chaining $\mathbf{E}[t] = O(1)$ $\mathbf{E}[t^k] = O(1)$ $t = O\left(\frac{\lg n}{\lg \lg n}\right)$ w.h.p.	2 $2k+1$ $\Theta\left(\frac{\lg n}{\lg \lg n}\right)$	
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.	$O\left(\lg \frac{1}{\varepsilon}\right)$ [Indyk'99]	$\Omega\left(\lg \frac{1}{\varepsilon}\right)$ **[PT ICALP'10]**

Independence has been the ruling measure for quality of hash functions for 30+ years, but is it right?

Simple Tabulation Hashing [Zobrist'70 chess]

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

Simple Tabulation Hashing [Zobrist'70 chess]

- Key x divided into $c = O(1)$ characters $x_1, ..., x_c$, e.g., 32-bit key as 4×8-bit characters.
- For $i = 1, ..., c$, we have truly random hash table:

  R_i : char $\rightarrow$ hash values (bit strings)

Simple Tabulation Hashing [Zobrist'70 chess]

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

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

- Space $cu^{1/c}$ and time $O(c)$.

Simple Tabulation Hashing [Zobrist'70 chess]

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

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

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

Simple Tabulation Hashing [Zobrist'70 chess]

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

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

- Space $cu^{1/c}$ and time $O(c)$.
- With 8-bit characters, each R_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 $x_1, ..., x_c$, e.g., 32-bit key as 4×8-bit characters.
- For $i = 1, ..., c$, we have truly random hash table:
  R_i : char $\rightarrow$ hash values (bit strings)
- Hash value

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

- Space $cu^{1/c}$ and time $O(c)$.
- With 8-bit characters, each R_i has 256 entries and fit in L1 cache.
- Simple tabulation is the fastest 3-independent hashing scheme. Speed like 2 multiplications.
- Not 4-independent: $h(a_1 a_2) \oplus h(a_1 b_2) \oplus h(b_1 a_2) \oplus h(b_1 b_2)$

$$\begin{aligned} = \quad & (R_1[a_1] \oplus R_2[a_2]) \oplus (R_1[a_1] \oplus R_2[b_2]) \oplus \\ & (R_1[b_1] \oplus R_2[a_2]) \oplus (R_1[b_1] \oplus R_2[b_2]) \; = \; 0. \end{aligned}$$

How much independence needed? Wrong question

Chaining $\mathbf{E}[t] = O(1)$ $\mathbf{E}[t^k] = O(1)$ $t = O(\frac{\lg n}{\lg \lg n})$ w.h.p.	2 $2k+1$ $\Theta(\frac{\lg n}{\lg \lg n})$	
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]**

How much independence needed? Wrong question

Chaining $\mathbf{E}[t] = O(1)$ $\mathbf{E}[t^k] = O(1)$ $t = O(\frac{\lg n}{\lg \lg n})$ w.h.p.	2 $2k+1$ $\Theta(\frac{\lg n}{\lg \lg n})$	
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]**

New result: Despite its 4-dependence, simple tabulation suffices for all the above applications:

> *One simple and fast hashing scheme for almost all your needs.*

How much independence needed? Wrong question

Chaining $\mathbf{E}[t] = O(1)$ $\mathbf{E}[t^k] = O(1)$ $t = O(\frac{\lg n}{\lg \lg n})$ w.h.p.	2 $2k+1$ $\Theta(\frac{\lg n}{\lg \lg n})$	
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]**

New result: Despite its 4-dependence, simple tabulation suffices for all the above applications:

> *One simple and fast hashing scheme for almost all your needs.*

Knuth recommends simple tabulation but cites only 3-independence as mathematical quality.

How much independence needed? Wrong question

Chaining $\mathbf{E}[t] = O(1)$ $\mathbf{E}[t^k] = O(1)$ $t = O(\frac{\lg n}{\lg \lg n})$ w.h.p.	2 $2k+1$ $\Theta(\frac{\lg n}{\lg \lg n})$	
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]**

New result: Despite its 4-dependence, simple tabulation suffices for all the above applications:

> *One simple and fast hashing scheme for almost all your needs.*

Knuth recommends simple tabulation but cites only 3-independence as mathematical quality.
We prove that dependence of simple tabulation is not harmful in any of the above applications.

Chaining/hashing into bins

Theorem Consider hashing n balls into $m \geq n^{1-1/(2c)}$ bins by simple tabulation. Let q be an additional *query ball*, and define X_q as the number of regular balls that hash into a bin chosen as a function of $h(q)$. Let $\mu = \mathbf{E}[X_q] = \frac{n}{m}$. The following probability bounds hold for any constant γ:

$$\Pr[X_q \geq (1+\delta)\mu] \leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\Omega(\mu)} + m^{-\gamma}$$

$$\Pr[X_q \leq (1-\delta)\mu] \leq \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{\Omega(\mu)} + m^{-\gamma}$$

With $m \leq n$ bins, every bin gets

$$n/m \pm O\left(\sqrt{n/m}\log^c n\right).$$

keys with probability $1 - n^{-\gamma}$.

Hashing into many bins

Lemma If we hash n keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

Hashing into many bins

Lemma If we hash n keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

Nothing like this lemma holds if we instead of simple tabulation assumed k-independent hashing with $k = O(1)$.

Hashing into many bins

Lemma If we hash n keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.
Proof that for any positive constants ε, γ, if we hash n keys into m bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.

Hashing into many bins

Lemma If we hash n keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

Proof that for any positive constants ε, γ, if we hash n keys into m bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.

Claim 1 Any set T contains a subset U of $\log_2 |T|$ keys that hash independently.

Hashing into many bins

Lemma If we hash n keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.
Proof that for any positive constants ε, γ, if we hash n keys into m bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.
Claim 1 Any set T contains a subset U of $\log_2 |T|$ keys that hash independently.

- Let i be character position where keys in T differ.

Hashing into many bins

Lemma If we hash n keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

Proof that for any positive constants ε, γ, if we hash n keys into m bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.

Claim 1 Any set T contains a subset U of $\log_2 |T|$ keys that hash independently.

- Let i be character position where keys in T differ.
- Let a be least common character in position i and pick $x \in T$ with $x_i = a$

Hashing into many bins

Lemma If we hash n keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

Proof that for any positive constants ε, γ, if we hash n keys into m bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.

Claim 1 Any set T contains a subset U of $\log_2 |T|$ keys that hash independently.

- Let i be character position where keys in T differ.
- Let a be least common character in position i and pick $x \in T$ with $x_i = a$
- Reduce T to T' removing all keys y from T with $y_i = a$.

Hashing into many bins

Lemma If we hash n keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

Proof that for any positive constants ε, γ, if we hash n keys into m bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.

Claim 1 Any set T contains a subset U of $\log_2 |T|$ keys that hash independently.

- Let i be character position where keys in T differ.
- Let a be least common character in position i and pick $x \in T$ with $x_i = a$
- Reduce T to T' removing all keys y from T with $y_i = a$.
- The hash of x is independent of the hash of T' as only $h(x)$ depends on $R_i[a]$.

Hashing into many bins

Lemma If we hash n keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.
Proof that for any positive constants ε, γ, if we hash n keys into m bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.
Claim 1 Any set T contains a subset U of $\log_2 |T|$ keys that hash independently.

- Let i be character position where keys in T differ.
- Let a be least common character in position i and pick $x \in T$ with $x_i = a$
- Reduce T to T' removing all keys y from T with $y_i = a$.
- The hash of x is independent of the hash of T' as only $h(x)$ depends on $R_i[a]$.
- Return $\{x\} \cup U'$ where U' independent subset of T'.

Hashing into many bins

Lemma If we hash n keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.
Proof that for any positive constants ε, γ, if we hash n keys into m bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.
Claim 1 Any set T contains a subset U of $\log_2 |T|$ keys that hash independently—if $|T| \geq d$ then $|U| \geq (1+\gamma)/\varepsilon$. $\square$

Hashing into many bins

Lemma If we hash n keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

Proof that for any positive constants ε, γ, if we hash n keys into m bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.

Claim 1 Any set T contains a subset U of $\log_2 |T|$ keys that hash independently—if $|T| \geq d$ then $|U| \geq (1+\gamma)/\varepsilon$. $\square$

Claim 2 The probability that there exists $u = (1+\gamma)/\varepsilon$ keys hashing independently to the same bin is $m^{-\gamma}$.

Hashing into many bins

Lemma If we hash n keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

Proof that for any positive constants ε, γ, if we hash n keys into m bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.

Claim 1 Any set T contains a subset U of $\log_2 |T|$ keys that hash independently—if $|T| \geq d$ then $|U| \geq (1+\gamma)/\varepsilon$. $\square$

Claim 2 The probability that there exists $u = (1+\gamma)/\varepsilon$ keys hashing independently to the same bin is $m^{-\gamma}$.

- At most $\binom{n}{u} < n^u$ independent sets U of u keys to consider.

Hashing into many bins

Lemma If we hash n keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.
Proof that for any positive constants ε, γ, if we hash n keys into m bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.
Claim 1 Any set T contains a subset U of $\log_2 |T|$ keys that hash independently—if $|T| \geq d$ then $|U| \geq (1+\gamma)/\varepsilon$. $\square$
Claim 2 The probability that there exists $u = (1+\gamma)/\varepsilon$ keys hashing independently to the same bin is $m^{-\gamma}$.

- At most $\binom{n}{u} < n^u$ independent sets U of u keys to consider.
- Each such U hash to *one* bin with probability $1/m^{u-1}$.

Hashing into many bins

Lemma If we hash n keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.
Proof that for any positive constants ε, γ, if we hash n keys into m bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.
Claim 1 Any set T contains a subset U of $\log_2 |T|$ keys that hash independently—if $|T| \geq d$ then $|U| \geq (1+\gamma)/\varepsilon$. $\square$
Claim 2 The probability that there exists $u = (1+\gamma)/\varepsilon$ keys hashing independently to the same bin is $m^{-\gamma}$.

- At most $\binom{n}{u} < n^u$ independent sets U of u keys to consider.
- Each such U hash to *one* bin with probability $1/m^{u-1}$.
- Propability bound over all U is

$$n^u m^{u-1} \leq m^{(1-\varepsilon)u+1-u} = m^{1-\varepsilon u} = m^{-\gamma}.$$

Hashing into many bins

Lemma If we hash n keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.

Proof that for any positive constants ε, γ, if we hash n keys into m bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.

Claim 1 Any set T contains a subset U of $\log_2 |T|$ keys that hash independently—if $|T| \geq d$ then $|U| \geq (1+\gamma)/\varepsilon$. $\square$

Claim 2 The probability that there exists $u = (1+\gamma)/\varepsilon$ keys hashing independently to the same bin is $m^{-\gamma}$. $\square$

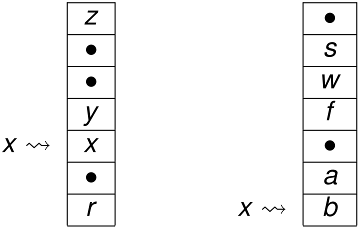
Hashing into many bins

Lemma If we hash n keys into $n^{1+\Omega(1)}$ bins, then all bins get $O(1)$ keys w.h.p.
Proof that for any positive constants ε, γ, if we hash n keys into m bins and $n \leq m^{1-\varepsilon}$, then all bins get less than $d = 2^{(1+\gamma)/\varepsilon}$ keys with probability $\geq 1 - m^{-\gamma}$.
Claim 1 Any set T contains a subset U of $\log_2 |T|$ keys that hash independently—if $|T| \geq d$ then $|U| \geq (1+\gamma)/\varepsilon$. $\square$
Claim 2 The probability that there exists $u = (1+\gamma)/\varepsilon$ keys hashing independently to the same bin is $m^{-\gamma}$.$\square$

Fundamental point Simple tabulation is only 3-independent, but inside any set T of size $\omega(1)$, there is a subset $S \subseteq T$ of size $\omega(1)$ where all keys in S are hashed independently.

Cuckoo hashing

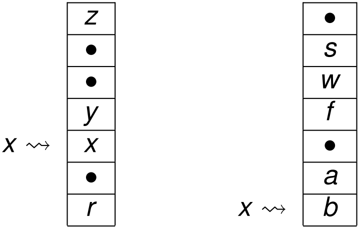
Each key placed in one of two hash locations.



Theorem With simple tabulation Cuckoo hashing works with probability $1 - \tilde{\Theta}(n^{-1/3})$.

Cuckoo hashing

Each key placed in one of two hash locations.

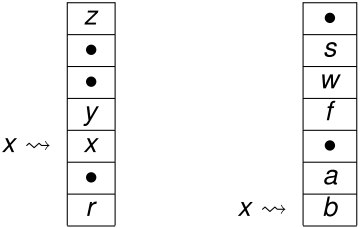


Theorem With simple tabulation Cuckoo hashing works with probability $1 - \tilde{\Theta}(n^{-1/3})$.

- For chaining and linear probing, we did not care about a constant loss, but obstructions to cuckoo hashing may be of just constant size, e.g., 3 keys sharing same two hash locations.

Cuckoo hashing

Each key placed in one of two hash locations.



Theorem With simple tabulation Cuckoo hashing works with probability $1 - \tilde{\Theta}(n^{-1/3})$.

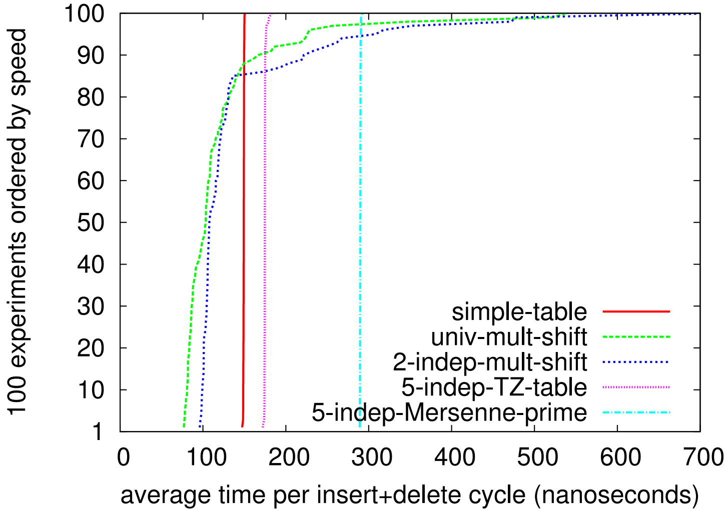
- For chaining and linear probing, we did not care about a constant loss, but obstructions to cuckoo hashing may be of just constant size, e.g., 3 keys sharing same two hash locations.
- Very delicate proof showing that obstruction can be used to code random tables R_i with few bits.

Speed

Hashing random keys		32-bit computer	64-bit computer
bits	hashing scheme	hashing time (ns)	
32	univ-mult-shift `(a*x)>>s`	1.87	2.33
32	2-indep-mult-shift	5.78	2.88
32	5-indep-Mersenne-prime	99.70	45.06
32	5-indep-TZ-table	10.12	12.66
32	simple-table	4.98	4.61
64	univ-mult-shift	7.05	3.14
64	2-indep-mult-shift	22.91	5.90
64	5-indep-Mersenne-prime	241.99	68.67
64	5-indep-TZ-table	75.81	59.84
64	simple-table	15.54	11.40

Experiments with help from Yin Zhang.

Robustness in linear probing for dense interval



Pitch for theory in case of linear probing

- Multiplicative hashing used in practice, but turns out to be very unreliable under typical denial-of-service (DoS) attacks based on consecutive IP addresses: systematic good performance 95% of the time, but systematic terrible performance 5% of the time [TZ'10].

Pitch for theory in case of linear probing

- Multiplicative hashing used in practice, but turns out to be very unreliable under typical denial-of-service (DoS) attacks based on consecutive IP addresses: systematic good performance 95% of the time, but systematic terrible performance 5% of the time [TZ'10].
- Problems in randomized algorithms like hashing hard to detect for practitioners. Hard for them to know if bad performance is from being unlucky, or because of systematic problems.

Pitch for theory in case of linear probing

- Multiplicative hashing used in practice, but turns out to be very unreliable under typical denial-of-service (DoS) attacks based on consecutive IP addresses: systematic good performance 95% of the time, but systematic terrible performance 5% of the time [TZ'10].
- Problems in randomized algorithms like hashing hard to detect for practitioners. Hard for them to know if bad performance is from being unlucky, or because of systematic problems.
- Linear probing had gotten a reputation for being fastest in practice, but sometimes unreliable needing special protection against bad cases.

Pitch for theory in case of linear probing

- Multiplicative hashing used in practice, but turns out to be very unreliable under typical denial-of-service (DoS) attacks based on consecutive IP addresses: systematic good performance 95% of the time, but systematic terrible performance 5% of the time [TZ'10].
- Problems in randomized algorithms like hashing hard to detect for practitioners. Hard for them to know if bad performance is from being unlucky, or because of systematic problems.
- Linear probing had gotten a reputation for being fastest in practice, but sometimes unreliable needing special protection against bad cases.
- Here we proved linear probing safe with good probabilistic performance for all input if we use simple tabulation.

Pitch for theory in case of linear probing

- Multiplicative hashing used in practice, but turns out to be very unreliable under typical denial-of-service (DoS) attacks based on consecutive IP addresses: systematic good performance 95% of the time, but systematic terrible performance 5% of the time [TZ'10].
- Problems in randomized algorithms like hashing hard to detect for practitioners. Hard for them to know if bad performance is from being unlucky, or because of systematic problems.
- Linear probing had gotten a reputation for being fastest in practice, but sometimes unreliable needing special protection against bad cases.
- Here we proved linear probing safe with good probabilistic performance for all input if we use simple tabulation.
- Simple tabulation also powerful for chaining, cuckoo hashing, and min-wise hashing:

  one simple and fast scheme for (almost) all your needs.

Twisted Tabulation

- With chaining and linear probing, each operation takes expected constant time, but out of $\sqrt{n}$ operations, some are expected to take $\tilde{\Omega}(\log n)$ time.

Twisted Tabulation

- With chaining and linear probing, each operation takes expected constant time, but out of $\sqrt{n}$ operations, some are expected to take $\tilde{\Omega}(\log n)$ time.
- With truly random hash function, we handle every window of $\log n$ operations in $O(\log n)$ time w.h.p.

Twisted Tabulation

- With chaining and linear probing, each operation takes expected constant time, but out of $\sqrt{n}$ operations, some are expected to take $\tilde{\Omega}(\log n)$ time.
- With truly random hash function, we handle every window of $\log n$ operations in $O(\log n)$ time w.h.p.
- Hence, with small buffer (as in Internet routers), we do get down to constant time per operation!

Twisted Tabulation

- With chaining and linear probing, each operation takes expected constant time, but out of $\sqrt{n}$ operations, some are expected to take $\tilde{\Omega}(\log n)$ time.
- With truly random hash function, we handle every window of $\log n$ operations in $O(\log n)$ time w.h.p.
- Hence, with small buffer (as in Internet routers), we do get down to constant time per operation!
- Simple tabulation does not achieve this: may often spend $\tilde{\Omega}(\log^2 n)$ time on $\log n$ consecutive operations.

Twisted Tabulation

- With chaining and linear probing, each operation takes expected constant time, but out of $\sqrt{n}$ operations, some are expected to take $\tilde{\Omega}(\log n)$ time.
- With truly random hash function, we handle every window of $\log n$ operations in $O(\log n)$ time w.h.p.
- Hence, with small buffer (as in Internet routers), we do get down to constant time per operation!
- Simple tabulation does not achieve this: may often spend $\tilde{\Omega}(\log^2 n)$ time on $\log n$ consecutive operations.
- Twisted tabulation:

$$
\begin{aligned}
(h, \alpha) &= R_1[x_1] \oplus \cdots \oplus R_{c-1}[x_{c-1}]; \\
h(x) &= h \oplus R_c[\alpha \oplus x_c]
\end{aligned}
$$

Twisted Tabulation

- With chaining and linear probing, each operation takes expected constant time, but out of $\sqrt{n}$ operations, some are expected to take $\tilde{\Omega}(\log n)$ time.
- With truly random hash function, we handle every window of $\log n$ operations in $O(\log n)$ time w.h.p.
- Hence, with small buffer (as in Internet routers), we do get down to constant time per operation!
- Simple tabulation does not achieve this: may often spend $\tilde{\Omega}(\log^2 n)$ time on $\log n$ consecutive operations.
- Twisted tabulation:

$$\begin{aligned}(h, \alpha) &= R_1[x_1] \oplus \cdots \oplus R_{c-1}[x_{c-1}];\\ h(x) &= h \oplus R_c[\alpha \oplus x_c]\end{aligned}$$

- With twisted tabulation, we handle every window of $\log n$ operations in $O(\log n)$ time w.h.p.

C-code for twisted tabulation (32-bit key, 8-bit char)

```
INT32 TwistedTab32(INT32 x, INT64[4][256] H) {
  INT32 i; INT64 h=0; INT8 c;
  for (i=0;i<c;i++) {
    c=x;
    h^=H[i][c];
    x = x>> 8;
  }
  c=x^h;        //twisted character
  h^=H[i][c];
  h>>=8;        //dropping twister from hash
  return ((INT32) h);
}
```

General Chernoff bounds with twisted tabulation

- 0-1 variables X_i where $X_i = 1$ with probability p_i.

General Chernoff bounds with twisted tabulation

- 0-1 variables X_i where $X_i = 1$ with probability p_i.
- Exponential concentration of $X = \sum_i X_i$ around mean.

General Chernoff bounds with twisted tabulation

- 0-1 variables X_i where $X_i = 1$ with probability p_i.
- Exponential concentration of $X = \sum_i X_i$ around mean.
- Application: trust polynomial number of logarithmic estimates with high probability
  —the log factor in many randomized algorithms.

General Chernoff bounds with twisted tabulation

- 0-1 variables X_i where $X_i = 1$ with probability p_i.
- Exponential concentration of $X = \sum_i X_i$ around mean.
- Application: trust polynomial number of logarithmic estimates with high probability
  —the log factor in many randomized algorithms.
- With hashing into $[0, 1]$, set $X_i = 1$ if $h(i) < p_i$.

General Chernoff bounds with twisted tabulation

- 0-1 variables X_i where $X_i = 1$ with probability p_i.
- Exponential concentration of $X = \sum_i X_i$ around mean.
- Application: trust polynomial number of logarithmic estimates with high probability
  —the log factor in many randomized algorithms.
- With hashing into $[0, 1]$, set $X_i = 1$ if $h(i) < p_i$.
- With bounded independence only polynomial concentration.

General Chernoff bounds with twisted tabulation

- 0-1 variables X_i where $X_i = 1$ with probability p_i.
- Exponential concentration of $X = \sum_i X_i$ around mean.
- Application: trust polynomial number of logarithmic estimates with high probability
  —the log factor in many randomized algorithms.
- With hashing into $[0, 1]$, set $X_i = 1$ if $h(i) < p_i$.
- With bounded independence only polynomial concentration.
- With twisted tabulation: for any constant γ,

$$\Pr[X \geq (1+\delta)\mu] \leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\Omega(\mu)} + u^{-\gamma}$$

$$\Pr[X \leq (1-\delta)\mu] \leq \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{\Omega(\mu)} + u^{-\gamma}$$

General Chernoff bounds with twisted tabulation

- 0-1 variables X_i where $X_i = 1$ with probability p_i.
- Exponential concentration of $X = \sum_i X_i$ around mean.
- Application: trust polynomial number of logarithmic estimates with high probability
  —the log factor in many randomized algorithms.
- With hashing into $[0, 1]$, set $X_i = 1$ if $h(i) < p_i$.
- With bounded independence only polynomial concentration.
- With twisted tabulation: for any constant γ,

$$\Pr[X \geq (1+\delta)\mu] \leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\Omega(\mu)} + u^{-\gamma}$$

$$\Pr[X \leq (1-\delta)\mu] \leq \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{\Omega(\mu)} + u^{-\gamma}$$

- With simple tabulation, additive term $(\max_i p_i)^{\gamma}$
  —in the hash tables we had $p \approx 1/n$.

Min-wise hashing with Søren Dahlgaard

- Min-wise hashing h of keys in U with bias ε:

$$x \in S \subseteq U : \qquad \Pr[h(x) = \min h(S)] = \frac{1 \pm \varepsilon}{|S|}$$

Min-wise hashing with Søren Dahlgaard

- Min-wise hashing h of keys in U with bias ε:

$$x \in S \subseteq U : \qquad \Pr[h(x) = \min h(S)] = \frac{1 \pm \varepsilon}{|S|}$$

- With simple tabulation, we get $\varepsilon = \tilde{O}(1/|S|^{1/c})$ — good only for large sets S.

Min-wise hashing with Søren Dahlgaard

- Min-wise hashing h of keys in U with bias ε:

$$x \in S \subseteq U: \qquad \Pr[h(x) = \min h(S)] = \frac{1 \pm \varepsilon}{|S|}$$

- With simple tabulation, we get $\varepsilon = \tilde{O}(1/|S|^{1/c})$
— good only for large sets S.
- With twisted tabulation, we get $\varepsilon = \tilde{O}(1/|U|^{1/c})$.

Speed of different schemes

Hashing scheme	time (ns)
2-indep: multiplication-shift [Dietzfelbinger'96]	1.27
3-indep: Polynomial with Mersenne prime $2^{61}-1$	7.72
5-indep: Polynomial with Mersenne prime $2^{61}-1$	14.09
k-indep: Polynomial with Mersenne prime $2^{61}-1$	$\approx 3.5(k-1)$
simple tabulation	2.26
twisted tabulation	2.86

Speed of different schemes

Hashing scheme	time (ns)
2-indep: multiplication-shift [Dietzfelbinger'96]	1.27
3-indep: Polynomial with Mersenne prime $2^{61}-1$	7.72
5-indep: Polynomial with Mersenne prime $2^{61}-1$	14.09
k-indep: Polynomial with Mersenne prime $2^{61}-1$	$\approx 3.5(k-1)$
simple tabulation	2.26
twisted tabulation	2.86
Pseudo-random number generator	time (ns)
random() from C-libraray	6.99
twisted-PRG()	1.18

Open problems for Simple and Twisted Tabulation

- Take any application using abstract truly random hash function, and prove that simple/twisted tabulation works.

Open problems for Simple and Twisted Tabulation

- Take any application using abstract truly random hash function, and prove that simple/twisted tabulation works.
- Could this be the first implementable hash function/RNG making classic quick sort work directly: using hash of i to generate index of ith pivot?

Open problems for Simple and Twisted Tabulation

- Take any application using abstract truly random hash function, and prove that simple/twisted tabulation works.
- Could this be the first implementable hash function/RNG making classic quick sort work directly: using hash of i to generate index of ith pivot?
- Recently with Søren Dahlgaar, Mathias Bæk Tejs Knudsen, and Eva Rotenberg: simple tabulation yields good results for power of two choices, e.g., max load $O(\log \log n)$ w.h.p., placing n keys in $\Omega(n)$ bins (each key hash to two bins and goes for the lighter loaded).

Double Tabulation

Recall that simple tabulation is only 3-independent (and so is twisted).

Double Tabulation

Recall that simple tabulation is only 3-independent (and so is twisted).
But if you really want high independence, then just apply simple tabulation twice...

k-independence [Wegman Carter FOCS'79]

Hashing universe U of keys into range R of hash values.

Random hash function $h : U \to R$ is k-independent iff:

- $(\forall) x \in U$, $h(x)$ is (almost) uniform in R;
- $(\forall) x_1, \ldots, x_k \in U$, $h(x_1), \ldots, h(x_k)$ are independent.

Prototypical example: Polynomial over prime field $\mathbb{Z}_p$:

- choose $a_0, a_1, \ldots, a_{k-1}$ randomly in $\mathbb{Z}_p$;
- $h(x) = \left(a_0 + a_1 x + \cdots + a_{k-1} x^{k-1}\right) \bmod p$.
- then $h : \mathbb{Z}_p \to \mathbb{Z}_p$ is k-independent with exact uniformity.

Ideal when it comes to space and amount of randomness:

- store k random elements from $\mathbb{Z}_p$,

k-independence [Wegman Carter FOCS'79]

Hashing universe U of keys into range R of hash values.

Random hash function $h : U \to R$ is k-independent iff:

- $(\forall)x \in U$, $h(x)$ is (almost) uniform in R;
- $(\forall)x_1, \ldots, x_k \in U$, $h(x_1), \ldots, h(x_k)$ are independent.

Prototypical example: Polynomial over prime field $\mathbb{Z}_p$:

- choose $a_0, a_1, \ldots, a_{k-1}$ randomly in $\mathbb{Z}_p$;
- $h(x) = \left(a_0 + a_1 x + \cdots + a_{k-1} x^{k-1}\right) \bmod p$.
- then $h : \mathbb{Z}_p \to \mathbb{Z}_p$ is k-independent with exact uniformity.

Ideal when it comes to space and amount of randomness:

- store k random elements from $\mathbb{Z}_p$,
- but evaluating $h(x)$ takes $O(k)$ time.

Siegel on highly independence [FOCS'89]

Polynomials: k-independent hashing in k time using k space

Question Can we get k-independent hashing in $t < k$ time?

Siegel on highly independence [FOCS'89]

Polynomials: k-independent hashing in k time using k space

- **Question** Can we get k-independent hashing in $t < k$ time?
- **Negative** $t < k$ time (cell probes) requires $|U|^{1/t}$ space.

Siegel on highly independence [FOCS'89]

Polynomials: k-independent hashing in k time using k space

Question Can we get k-independent hashing in $t < k$ time?

Negative $t < k$ time (cell probes) requires $|U|^{1/t}$ space.

Positive $|U|^{\Omega(1/c^2)}$-independence in $O(c)^c$ time using $|U|^{1/c}$ space.

Siegel on highly independence [FOCS'89]

Polynomials: k-independent hashing in k time using k space

- **Question** Can we get k-independent hashing in $t < k$ time?
- **Negative** $t < k$ time (cell probes) requires $|U|^{1/t}$ space.
- **Positive** $|U|^{\Omega(1/c^2)}$-independence in $O(c)^c$ time using $|U|^{1/c}$ space.

Siegel's formulation With $t, c = O(1)$:

- **Negative** $\omega(1)$-independence in $O(1)$ time requires $|U|^{\Omega(1)}$ space.

Siegel on highly independence [FOCS'89]

Polynomials: k-independent hashing in k time using k space

- **Question** Can we get k-independent hashing in $t < k$ time?
- **Negative** $t < k$ time (cell probes) requires $|U|^{1/t}$ space.
- **Positive** $|U|^{\Omega(1/c^2)}$-independence in $O(c)^c$ time using $|U|^{1/c}$ space.

Siegel's formulation With $t, c = O(1)$:

- **Negative** $\omega(1)$-independence in $O(1)$ time requires $|U|^{\Omega(1)}$ space.
- **Positive** $|U|^{\Omega(1)}$-independence in $O(1)$ time using $|U|^{\varepsilon}$ space for any $\varepsilon = \Omega(1)$.

Siegel on highly independence [FOCS'89]

Polynomials: k-independent hashing in k time using k space

Question Can we get k-independent hashing in $t < k$ time?

Negative $t < k$ time (cell probes) requires $|U|^{1/t}$ space.

Positive $|U|^{\Omega(1/c^2)}$-independence in $O(c)^c$ time using $|U|^{1/c}$ space.

Siegel's formulation With $t, c = O(1)$:

Negative $\omega(1)$-independence in $O(1)$ time requires $|U|^{\Omega(1)}$ space.

Positive $|U|^{\Omega(1)}$-independence in $O(1)$ time using $|U|^{\varepsilon}$ space for any $\varepsilon = \Omega(1)$.

"far too slow for any practical application".

Siegel on highly independence [FOCS'89]

Polynomials: k-independent hashing in k time using k space

Question: Can we get k-independent hashing in $t < k$ time?

Negative: $t < k$ time (cell probes) needs $|U|^{1/t}$ space.

Positive: $|U|^{\Omega(1/c^2)}$-independence in $O(c)^c$ time using $|U|^{1/c}$ space.

"far too slow for any practical application".

New results [FOCS'13]

1. $|U|^{\Omega(1/c^2)}$-independence in optimal $O(c)$ time using $|U|^{1/c}$ space.

Siegel on highly independence [FOCS'89]

Polynomials: k-independent hashing in k time using k space

Question Can we get k-independent hashing in $t < k$ time?

Negative $t < k$ time (cell probes) needs $|U|^{1/t}$ space.

Positive $|U|^{\Omega(1/c^2)}$-independence in $O(c)^c$ time using $|U|^{1/c}$ space.

"far too slow for any practical application".

New results [FOCS'13]

1. $|U|^{\Omega(1/c^2)}$-independence in optimal $O(c)$ time using $|U|^{1/c}$ space.
2. $|U|^{\Omega(1/c)}$-independence in $O(c^{\log c})$ time using $|U|^{1/c}$ space.

Siegel on highly independence [FOCS'89]

Polynomials: k-independent hashing in k time using k space

Question: Can we get k-independent hashing in $t < k$ time?

Negative: $t < k$ time (cell probes) needs $|U|^{1/t}$ space.

Positive: $|U|^{\Omega(1/c^2)}$-independence in $O(c)^c$ time using $|U|^{1/c}$ space.

"far too slow for any practical application".

New results [FOCS'13]

1. $|U|^{\Omega(1/c^2)}$-independence in optimal $O(c)$ time using $|U|^{1/c}$ space.
2. $|U|^{\Omega(1/c)}$-independence in $O(c^{\log c})$ time using $|U|^{1/c}$ space.

Schemes are simple but analysis is not.

Simple tabulation

- Key $x = (x_0, ..., x_{c-1}) \in \Phi^c = U$, $c = O(1)$.
- For $i = 0, , ..., c - 1$, truly random character hash table:
  $h_i : \Phi \to R = b$-bit strings.
- Hash function $h : U \to R$ defined by

$$h(x_0, ..., x_{c-1}) = h_0[x_0] \oplus \cdots \oplus h_{c-1}[x_{c-1}]$$

- We saw not 4-independent, yet powerful enough for many algorithmic contexts otherwise requiring higher independence.

Simple tabulation

- Key $x = (x_0, ..., x_{c-1}) \in \Phi^c = U$, $c = O(1)$.
- For $i = 0, , ..., c - 1$, truly random character hash table:
  $h_i : \Phi \to R = b$-bit strings.
- Hash function $h : U \to R$ defined by

$$h(x_0, ..., x_{c-1}) = h_0[x_0] \oplus \cdots \oplus h_{c-1}[x_{c-1}]$$

- We saw not 4-independent, yet powerful enough for many algorithmic contexts otherwise requiring higher independence.

We now claim that simple tabulation is expected to yield:

- Very fast unbalanced constant degree expanders.

Simple tabulation

- Key $x = (x_0, ..., x_{c-1}) \in \Phi^c = U$, $c = O(1)$.
- For $i = 0, , ..., c - 1$, truly random character hash table:
  $h_i : \Phi \to R = b$-bit strings.
- Hash function $h : U \to R$ defined by

$$h(x_0, ..., x_{c-1}) = h_0[x_0] \oplus \cdots \oplus h_{c-1}[x_{c-1}]$$

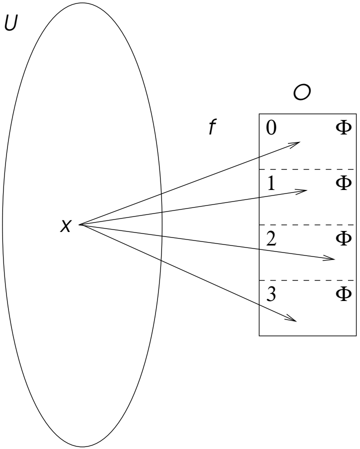
- We saw not 4-independent, yet powerful enough for many algorithmic contexts otherwise requiring higher independence.

We now claim that simple tabulation is expected to yield:

- Very fast unbalanced constant degree expanders.
- Applied twice yields highly independent hashing.

Simple tabulation as unbalanced expander

- Consider any function $f : U \to \Phi^d$.
- Defines unbalanced bipartite graph between keys U and "output position characters" in $O = [d] \times \Phi$.
- If $f(x) = (y_0, ..., y_{d-1})$ then $N_f(x) = \{(0, y_0), ...(d-1, y_{d-1})\}$.



Simple tabulation as unbalanced expander

- Consider any function $f : U \to \Phi^d$.
- Defines unbalanced bipartite graph between keys U and "output position characters" in $O = [d] \times \Phi$.
- If $f(x) = (y_0, ..., y_{d-1})$ then $N_f(x) = \{(0, y_0), ...(d-1, y_{d-1})\}$.

Simple tabulation as unbalanced expander

- Consider any function $f : U \to \Phi^d$.
- Defines unbalanced bipartite graph between keys U and "output position characters" in $O = [d] \times \Phi$.
- If $f(x) = (y_0, ..., y_{d-1})$ then $N_f(x) = \{(0, y_0), ...(d-1, y_{d-1})\}$.

Thm Let $h : \Phi^c \to \Phi^d$ random simple tabulation function with $d = 12c$. Then with probability $1 - o(1/|\Phi|)$,

$$\forall S \subseteq U, |S| \leq |\Phi|^{1/(5c)} : |N_h(S)| > d|S|/2.$$

Simple tabulation as unbalanced expander

- Consider any function $f : U \to \Phi^d$.
- Defines unbalanced bipartite graph between keys U and "output position characters" in $O = [d] \times \Phi$.
- If $f(x) = (y_0, ..., y_{d-1})$ then $N_f(x) = \{(0, y_0), ...(d-1, y_{d-1})\}$.

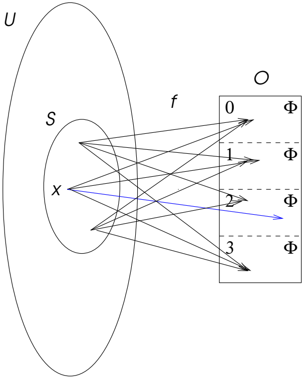
Thm Let $h : \Phi^c \to \Phi^d$ random simple tabulation function with $d = 12c$. Then with probability $1 - o(1/|\Phi|)$,

$$\forall S \subseteq U, |S| \leq |\Phi|^{1/(5c)} : |N_h(S)| > d|S|/2.$$

Note Best explicit unbalanced expanders [Guruswami et al. JACM'09] have logarithmic degrees, and would be orders of magnitude slower to compute.

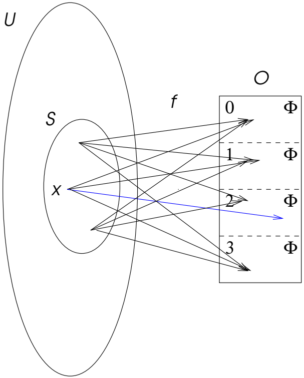
Unique output position character

Set $S \subseteq U$ has unique output position character if there is output position character neighbor to exactly one key in S.



Unique output position character

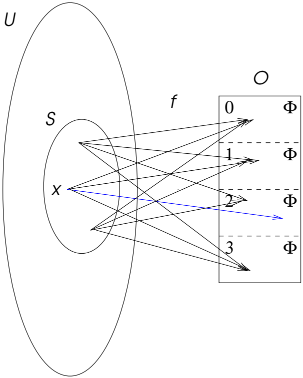
Set $S \subseteq U$ has unique output position character if there is output position character neighbor to exactly one key in S.



Obs S has a unique neighbor if $|N(S)| > d|S|/2$.

Unique output position character

Set $S \subseteq U$ has unique output position character if there is output position character neighbor to exactly one key in S.



Obs S has a unique neighbor if $|N(S)| > d|S|/2$.

Thm Let $h : \Phi^c \rightarrow \Phi^d$ random simple tabulation function with $d = 6c$. Then with probability $1 - o(1/|\Phi|)$,
$\forall S \subseteq U, |S| \leq |\Phi|^{1/(5c)}$: S has a unique output position character.

Unique output position character

A function $f : U \to \Phi^d$ is k-unique if

$$\forall S \subseteq U, |S| \leq k : S \text{ has a unique output position character .}$$

Thm $h : \Phi^c \to \Phi^d$ random simple tabulation function with $d = 6c$. Then h is $|\Phi|^{1/(5c)}$-unique with probability $1 - o(1/|\Phi|)$.

Unique output position character

A function $f : U \to \Phi^d$ is k-unique if

$$\forall S \subseteq U, |S| \leq k : S \text{ has a unique output position character }.$$

Thm $h : \Phi^c \to \Phi^d$ random simple tabulation function with $d = 6c$. Then h is $|\Phi|^{1/(5c)}$-unique with probability $1 - o(1/|\Phi|)$.

Lem (Siegel's peeling) If $f : U \to \Phi^d$ is k-unique and $r : \Phi^d \to R$ random simple tabulation function, then $r \circ f$ is k-independent.

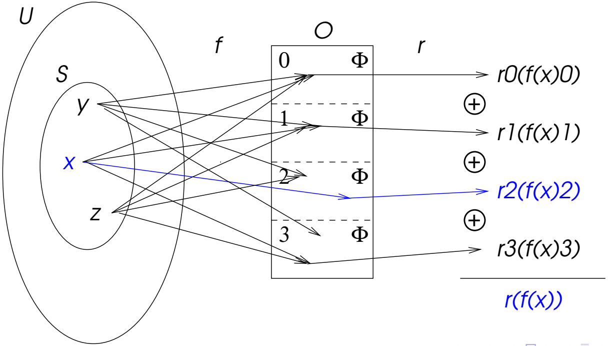
Unique output position character

A function $f : U \rightarrow \Phi^d$ is *k-unique* if

$$\forall S \subseteq U, |S| \leq k : S \text{ has a unique output position character .}$$

Thm $h : \Phi^c \rightarrow \Phi^d$ random simple tabulation function with $d = 6c$. Then h is $|\Phi|^{1/(5c)}$-unique with probability $1 - o(1/|\Phi|)$.

Lem (Siegel's peeling) If $f : U \rightarrow \Phi^d$ is k-unique and $r : \Phi^d \rightarrow R$ random simple tabulation function, then $r \circ f$ is k-independent.



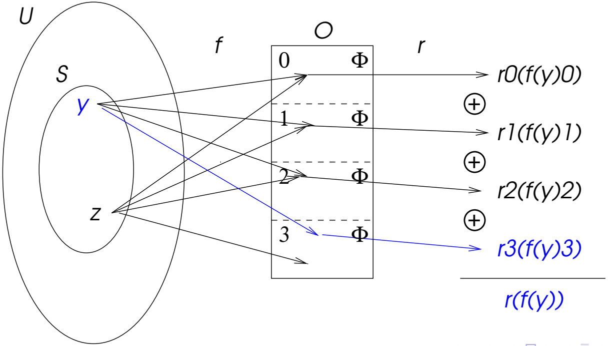
Unique output position character

A function $f : U \to \Phi^d$ is k-unique if

$$\forall S \subseteq U, |S| \leq k : S \text{ has a unique output position character.}$$

Thm $h : \Phi^c \to \Phi^d$ random simple tabulation function with $d = 6c$. Then h is $|\Phi|^{1/(5c)}$-unique with probability $1 - o(1/|\Phi|)$.

Lem (Siegel's peeling) If $f : U \to \Phi^d$ is k-unique and $r : \Phi^d \to R$ random simple tabulation function, then $r \circ f$ is k-independent.



Unique output position character

A function $f : U \to \Phi^d$ is *k-unique* if

$$\forall S \subseteq U, |S| \leq k : S \text{ has a unique output position character.}$$

Thm $h : \Phi^c \to \Phi^d$ random simple tabulation function with $d = 6c$. Then h is $|\Phi|^{1/(5c)}$-unique with probability $1 - o(1/|\Phi|)$.

Lem (Siegel's peeling) If $f : U \to \Phi^d$ is k-unique and $r : \Phi^d \to R$ random simple tabulation function, then $r \circ f$ is k-independent.

Cor (Double Tabulation) If $d = 6c$ and $h : U = \Phi^c \to \Phi^d$ and $r : \Phi^d \to R$ are random simple tabulation functions, then $r \circ h : U \to R$ is $|\Phi|^{1/(5c)}$-independent with "universal" probability $1 - o(1/|\Phi|)$.

Unique output position character

A function $f : U \to \Phi^d$ is k-unique if

$$\forall S \subseteq U, |S| \leq k : S \text{ has a unique output position character.}$$

Thm $h : \Phi^c \to \Phi^d$ random simple tabulation function with $d = 6c$. Then h is $|\Phi|^{1/(5c)}$-unique with probability $1 - o(1/|\Phi|)$.

Lem (Siegel's peeling) If $f : U \to \Phi^d$ is k-unique and $r : \Phi^d \to R$ random simple tabulation function, then $r \circ f$ is k-independent.

Cor (Double Tabulation) If $d = 6c$ and $h : U = \Phi^c \to \Phi^d$ and $r : \Phi^d \to R$ are random simple tabulation functions, then $r \circ h : U \to R$ is $|\Phi|^{1/(5c)}$-independent with "universal" probability $1 - o(1/|\Phi|)$.

Note A k-unique h can be used as a universal constant that only has to be guessed or constructed once.

With real constants

Constants in construction pretty good:

Thm For 32-bit keys divided in $c = 2$ characters from $\Phi = [2^{16}]$, if $h : \Phi^2 \to \Phi^{20}$ is random simple tabulation function, then h is 100-unique with probability $1 - 1.5 \times 10^{-42}$.

With real constants

Constants in construction pretty good:

Thm For 32-bit keys divided in $c = 2$ characters from $\Phi = [2^{16}]$, if $h : \Phi^2 \rightarrow \Phi^{20}$ is random simple tabulation function, then h is 100-unique with probability $1 - 1.5 \times 10^{-42}$.

Claiming that a USB flash drive with random bits represents a universal k-unique simple tabulation function, would be very safe hardware

Techniques

- The proof that a random simple tabulation hashing yields high expansion and a unique output position character for every set of size $\leq \Phi^{1/(5c)}$ is rather subtle.

Techniques

- The proof that a random simple tabulation hashing yields high expansion and a unique output position character for every set of size $\leq \Phi^{1/(5c)}$ is rather subtle.
- In violating set, keys share output position characters.

Techniques

- The proof that a random simple tabulation hashing yields high expansion and a unique output position character for every set of size $\leq \Phi^{1/(5c)}$ is rather subtle.
- In violating set, keys share output position characters.
- This leads to equations from which some output position characters can be derived efficiently. Hard part is to get enough independent equations.

Techniques

- The proof that a random simple tabulation hashing yields high expansion and a unique output position character for every set of size $\leq \Phi^{1/(5c)}$ is rather subtle.
- In violating set, keys share output position characters.
- This leads to equations from which some output position characters can be derived efficiently. Hard part is to get enough independent equations.
- The number of possible derivations is small compared to the number of possible output characters, so the event is unlikely with random output position characters.

Concluding remarks

- Siegel's highly independent hash functions have so far been a powerful theoretical tool, but ”far too slow for any practical application”.

Concluding remarks

- Siegel's highly independent hash functions have so far been a powerful theoretical tool, but ”far too slow for any practical application”.
- We presented a simple alternative which is exponentially faster and also yields very fast unbalanced expanders.

Concluding remarks

- Siegel's highly independent hash functions have so far been a powerful theoretical tool, but ”far too slow for any practical application”.
- We presented a simple alternative which is exponentially faster and also yields very fast unbalanced expanders.
- Simple, twisted, and double tabulation only needs pointers to randomly configured memory that does not need to be changed.

Concluding remarks

- Siegel's highly independent hash functions have so far been a powerful theoretical tool, but ”far too slow for any practical application”.
- We presented a simple alternative which is exponentially faster and also yields very fast unbalanced expanders.
- Simple, twisted, and double tabulation only needs pointers to randomly configured memory that does not need to be changed.
- In multi-core systems this leaves the cache clean with efficient concurrent processing.

Concluding remarks

- Siegel's highly independent hash functions have so far been a powerful theoretical tool, but "far too slow for any practical application".
- We presented a simple alternative which is exponentially faster and also yields very fast unbalanced expanders.
- Simple, twisted, and double tabulation only needs pointers to randomly configured memory that does not need to be changed.
- In multi-core systems this leaves the cache clean with efficient concurrent processing.
- One could even imagine using special cheap and fast randomly configured read only memory.

but do we still need high independence?

- Siegel's highly independent hashing is used as a sledge hammer in many algorithms (150+ citations).

but do we still need high independence?

- Siegel's highly independent hashing is used as a sledge hammer in many algorithms (150+ citations).
- Often easy to show that k independence suffices for an application, e.g., if analysis only involves events based on k hash values. One may later find alternatives, but double tabulation is a powerful first choice.

but do we still need high independence?

- Siegel's highly independent hashing is used as a sledge hammer in many algorithms (150+ citations).
- Often easy to show that k independence suffices for an application, e.g., if analysis only involves events based on k hash values. One may later find alternatives, but double tabulation is a powerful first choice.
- E.g., simple tabulation and twisted tabulation are both simpler and 20 times faster than double tabulation, and work for many applications.

but do we still need high independence?

- Siegel's highly independent hashing is used as a sledge hammer in many algorithms (150+ citations).
- Often easy to show that k independence suffices for an application, e.g., if analysis only involves events based on k hash values. One may later find alternatives, but double tabulation is a powerful first choice.
- E.g., simple tabulation and twisted tabulation are both simpler and 20 times faster than double tabulation, and work for many applications.
- For high probability Cuckoo hashing with a stash, and for uniform hashing, Dietzfelbinger et al. have found alternatives to Siegel's hashing, but double tabulation is a simple general replacement.

but do we still need high independence?

- Siegel's highly independent hashing is used as a sledge hammer in many algorithms (150+ citations).
- Often easy to show that k independence suffices for an application, e.g., if analysis only involves events based on k hash values. One may later find alternatives, but double tabulation is a powerful first choice.
- E.g., simple tabulation and twisted tabulation are both simpler and 20 times faster than double tabulation, and work for many applications.
- For high probability Cuckoo hashing with a stash, and for uniform hashing, Dietzfelbinger et al. have found alternatives to Siegel's hashing, but double tabulation is a simple general replacement.
- Double tabulation hashing also gives exponentially better error bounds in min-wise hashing and Chernoff bounds than those obtained with other hashing schemes.

High independence

Negative (Siegel) $t < k$ time (cell probes) needs $|U|^{1/t}$ space.

Positive new results

1. $|U|^{\Omega(1/c^2)}$-independent in optimal $O(c)$ time using $|U|^{1/c}$ space (double tabluation).
2. $|U|^{\Omega(1/c)}$-independence in $O(c^{\log c})$ time using $|U|^{1/c}$ space (recursive tabulation).

High independence

Negative (Siegel) $t < k$ time (cell probes) needs $|U|^{1/t}$ space.

Positive new results

1. $|U|^{\Omega(1/c^2)}$-independent in optimal $O(c)$ time using $|U|^{1/c}$ space (double tabluation).
2. $|U|^{\Omega(1/c)}$-independence in $O(c^{\log c})$ time using $|U|^{1/c}$ space (recursive tabulation).

2a. $|U|^{\Omega(1/c)}$-independence in $O(c \log c)$ time using $|U|^{1/c}$ space (differnt recursive tabulation with Tobias Lybecker Christiani and Rasmus Pagh).

High independence

Negative (Siegel) $t < k$ time (cell probes) needs $|U|^{1/t}$ space.

Positive new results

1. $|U|^{\Omega(1/c^2)}$-independent in optimal $O(c)$ time using $|U|^{1/c}$ space (double tabluation).

2a. $|U|^{\Omega(1/c)}$-independence in $O(c \log c)$ time using $|U|^{1/c}$ space (recursive tabulation with Tobias Lybecker Christiani and Rasmus Pagh).

High independence

Negative (Siegel) $t < k$ time (cell probes) needs $|U|^{1/t}$ space.

Positive new results

1. $|U|^{\Omega(1/c^2)}$-independent in optimal $O(c)$ time using $|U|^{1/c}$ space (double tabluation).

2a. $|U|^{\Omega(1/c)}$-independence in $O(c \log c)$ time using $|U|^{1/c}$ space (recursive tabulation with Tobias Lybecker Christiani and Rasmus Pagh).

Desired $|U|^{\Omega(1/c)}$-independence in optimal $O(c)$ time using $|U|^{1/c}$ space?

High independence

Negative (Siegel) $t < k$ time (cell probes) needs $|U|^{1/t}$ space.

Positive new results

1. $|U|^{\Omega(1/c^2)}$-independent in optimal $O(c)$ time using $|U|^{1/c}$ space (double tabluation).

2a. $|U|^{\Omega(1/c)}$-independence in $O(c \log c)$ time using $|U|^{1/c}$ space (recursive tabulation with Tobias Lybecker Christiani and Rasmus Pagh).

Desired $|U|^{\Omega(1/c)}$-independence in optimal $O(c)$ time using $|U|^{1/c}$ space?

It may be that double tabulation achieves this, but proving it would require different analysis.

High independence

Negative (Siegel) $t < k$ time (cell probes) needs $|U|^{1/t}$ space.

Positive new results

1. $|U|^{\Omega(1/c^2)}$-independent in optimal $O(c)$ time using $|U|^{1/c}$ space (double tabluation).

2a. $|U|^{\Omega(1/c)}$-independence in $O(c \log c)$ time using $|U|^{1/c}$ space (recursive tabulation with Tobias Lybecker Christiani and Rasmus Pagh).

Desired $|U|^{\Omega(1/c)}$-independence in optimal $O(c)$ time using $|U|^{1/c}$ space?

It may be that double tabulation achieves this, but proving it would require different analysis.

Recently With Søren Dahlgaar and Mathias Bæk Tejs Knudsen: for given set S of size $\leq |U|^{1/c}$, w.h.p., double tabulation truely random on S using $O(|U|^{1/c})$ space and $O(c)$ time.

Favorite open problem: do hash tables need hashing?

- Hash tables are used to look up keys in a dynamic set of stored keys. Can this be done without randomization?

Favorite open problem:
do hash tables need hashing?

- Hash tables are used to look up keys in a dynamic set of stored keys. Can this be done without randomization?
- Can we both insert and look up keys in constant deterministic time? (not just with high probability)

Favorite open problem: do hash tables need hashing?

- Hash tables are used to look up keys in a dynamic set of stored keys. Can this be done without randomization?
- Can we both insert and look up keys in constant deterministic time? (not just with high probability)
- Currently, the best answer is that we can do both in $O(\sqrt{\log n / \log\log n})$ worst-case time [Andersson Thorup STOC'00] —tight for more general predecessor problem.

Favorite open problem: do hash tables need hashing?

- Hash tables are used to look up keys in a dynamic set of stored keys. Can this be done without randomization?
- Can we both insert and look up keys in constant deterministic time? (not just with high probability)
- Currently, the best answer is that we can do both in $O(\sqrt{\log n / \log \log n})$ worst-case time [Andersson Thorup STOC'00] —tight for more general predecessor problem.
- Most people believe that deterministic constant time is not possible without randomization, but nobody can prove it.