On Randomness in Hash Functions

Martin Dietzfelbinger
Technische Universität Ilmenau

Co-Workers

- Andrea Montanari
- Andreas Goerdt
- Christoph Weidling
- Martin Aumüller
- Michael Mitzenmacher
- Michael Rink
- Rasmus Pagh
- Philipp Woelfel
- Ulf Schellbach

Outline

- **Part 1, The Cast:**
  Hash functions, hash classes, basic structures
  - Hash functions and classes
  - FRA: The full randomness assumption
  - d-wise independence versus ad-hoc-analysis
  - Simple tabulation
  - Tables plus one extra function
  - Key expansion and triangle property
  - Linear independence and d-wise independence

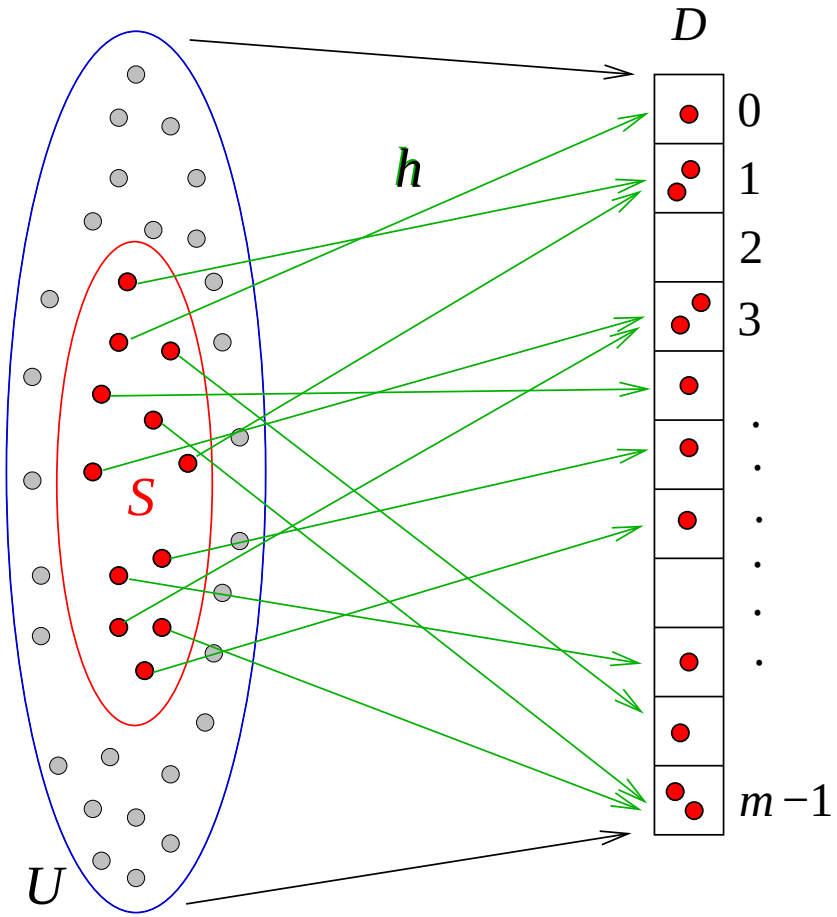
Outline

- **Part 2, The Scenes:**
  Applications and how they react to hash classes
  - Linear probing
  - Cuckoo hashing
  - . . . with a stash
  - Generalized cuckoo hashing
  - Simulation of uniform hashing
  - (Balanced allocation)
  - (Bloom filters)
  - (Retrieval: Representation of functions)

Part 1: The Cast

Hash Functions, Hash Classes, and their Abilities

Hash Functions



Hash Functions

Hash function:

$$h\colon U \to R.$$

U: "**universe**" of all "keys", $u = |U|$.

Often: $U = [u] = \{0, \ldots, u-1\}$ or $U = [c]^s$ ("words").

R: "**range**", size m, often $[m]$ or $\{0,1\}^\ell$.

A set $\boldsymbol{S} \subseteq U$: The set of interest (the keys that occur in the application at hand), $\boldsymbol{n} = |S|$.

Space requirements measured in relation to n.
($n^{1+\delta}$ is bad, $O(n)$ is tolerable [sometimes], n^δ is good.)

The Full Randomness Assumption

$\mathcal{H}_{\text{all}} = R^U = \{h \mid h \text{ is a hash function}\}$

"**random hash function**": the uniform distribution on $\mathcal{H}_{\text{all}}$.

Full Randomness Assumption (FRA):

We call "new function" and get a random $h \in \mathcal{H}_{\text{all}}$, which we can *store* and *evaluate* on $x \in U$.

Unrealistic (by information theory): Storing h needs $|U| \log |R|$ bits.

FRA "on S": space $n \log |R|$ bits.

Former textbook justification: **If** h distributes U evenly and $S \subseteq U$ is random then $(h(x_1), \ldots, h(x_n))$ is "almost" random in R^n.

Assumption "S is random" very questionable!

Classes of Hash Functions: Randomization

Idea **[Carter/Wegman 1977]**: "Universal hashing".

Put the randomness into the algorithm that computes the hash function.

A **"class of hash functions"** is a subset $\mathcal{H} \subseteq \mathcal{H}_{\text{all}}$, creating a probability space (uniform distribution on $\mathcal{H}$).

Usually given as **algorithm/formula** with **random parameters** that on input $x \in U$ calculates $h(x) \in R$.

"Randomized hash functions".

Classes of Hash Functions

E. g.: **Linear hash functions.** $U = [p] = \mathbb{Z}_p$ for a prime p.

$h(x) = ((\boldsymbol{a} \cdot x + \boldsymbol{b}) \bmod p) \bmod m.$ ($\boldsymbol{a}, \boldsymbol{b} \in [p]$ are random.)

Obvious generalization: **Higher degree polynomials** over $\mathbb{Z}_p$ with random coefficients (and "projection" into R).

E. g.: **Simple Tabulation.**

Key is a word/string $x = (c_1, \ldots, c_s)$ from $U = [u]^s$.

h described by: Tables $\boldsymbol{T_1}[0..u-1], \ldots, \boldsymbol{T_s}[0..u-1]$, filled with **random elements** of $(R, \oplus)$ (always: some **abelian group**)

$h((c_1, \ldots, c_s)) = \boldsymbol{T_1}[c_1] \oplus \ldots \oplus \boldsymbol{T_s}[c_s]$.

(Very efficient in practice.)

(d-Universality or) d-wise Independence

Definition **[Carter/Wegman 1977]**

Class $\mathcal{H} \subseteq \{h \mid h\colon U \to R\}$ is called ***d*-wise independent** if for arbitrary distinct keys $x_1, \ldots, x_d \in U$ and h chosen at random from $\mathcal{H}$ we have:

$(h(x_1), \ldots, h(x_d))$ is uniformly distributed in R^d,
i. e., the d hash values behave fully randomly.

Standard way of realizing d-wise independence:
$\mathcal{H} =$ the set of polynomials of degree up to $d-1$ over a finite field $\mathbb{F} = U$ (followed by some "projection" into R).

Extracting randomness from entropy in keys

A view at the FRA from a different angle.

We are dealing with (ordered) sets $S = \{x_1, \ldots, x_n\} \subseteq U$ of keys, "somewhat random".

[Mitzenmacher/Vadhan 2008], **[Chung/Vadhan 2008]**:

> **Some** randomness in keys + 2-wise independence
> = (close to) fully random hash values

Extracting randomness from entropy in keys

Theorem **[Mitzenmacher/Vadhan 2008]**

Assume $\varepsilon > 0$ and the distribution $\mathcal{D}$ that governs $(x_1, \ldots, x_n)$ is such that for each $i \in \{1, \ldots, n\}$ we have

$$\mathsf{cp}(x_i \mid x_1, \ldots, x_{i-1}) := \sum_{y \in U} \Pr_{\mathcal{D}}(x_i = y \mid x_1, \ldots, x_{i-1})^2 \leq \frac{\varepsilon^2}{|R|n},$$

("$\mathcal{D}$ is a block source with collision probability $\leq \varepsilon^2/(|R|n)$ per block").

Assume further that $h\colon U \to R$ is 2-wise independent.

Then the random vector $(h, h(x_1), \ldots, h(x_n))$ is ε-close to the uniform distribution on $\mathcal{H} \times R^n$.

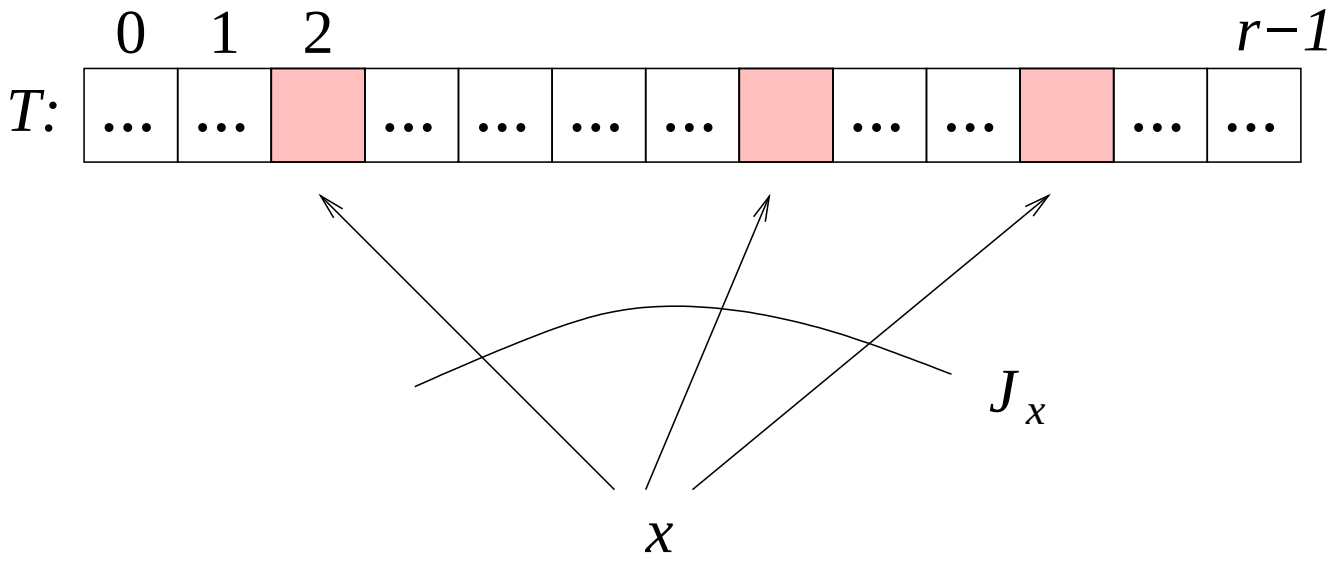
Extracting randomness from entropy in keys

This theorem **may explain** why weak (e. g., 2-wise independent) hash functions are so successful and work nicely in practice, but it does not give us the FRA for free:
One can identify "somewhat random" key sets S, often quite natural, sometimes somewhat construed, for which 2-wise independent classes behave badly.

Examples: Linear probing, cuckoo hashing (later).

General limitation: If the total entropy in the key set is smaller than $n \log |R|$, the hash values **can't be** close to random.

Recurring motif: Random table(s)



$$h(x) = \bigoplus_{j \in J_x} T[j].$$

Recurring motif: Random table(s)

Table $\boldsymbol{T}[0..r-1]$ of **random** elements from $(R, \oplus)$ (abelian group). Maybe a concatenation of several tables.

Typical: $r = n^\delta$ for constant $\delta < 1$.

Key $x \in U$ determines some entries in $T[0..r-1]$, given by index set $J_x \subseteq [r]$.

$$h(x) = \bigoplus_{j \in J_x} T[j].$$

Central feature: It's **fast** in practice.

Many variants, different methods of analysis.

A) Simple tabulation

Key is $x = (c_1, \ldots, c_s)$ from $U = [u_1] \times \ldots \times [u_s]$.

Use the s components as indices into s tables $\boldsymbol{T_1}[0..u_1 - 1], \ldots, \boldsymbol{T_s}[0..u_s - 1]$, entries from $(R, \oplus)$.

Easy: 3-wise independent, but not 4-wise independent.

Very strong randomness properties in several applications, like linear probing, simple cuckoo hashing, min-entropy, recognized only in **[Pătraşcu/Thorup 2011]**.

(These properties cannot be consequences of d-wise independence.)

B) Table plus d-wise independence (twice)

Two hash functions $\boldsymbol{f}: U \to R$ and $\boldsymbol{g}: U \to [r]$ from d-wise independent classes and a table $\boldsymbol{T}[0..r-1]$ over $(R, \oplus)$.

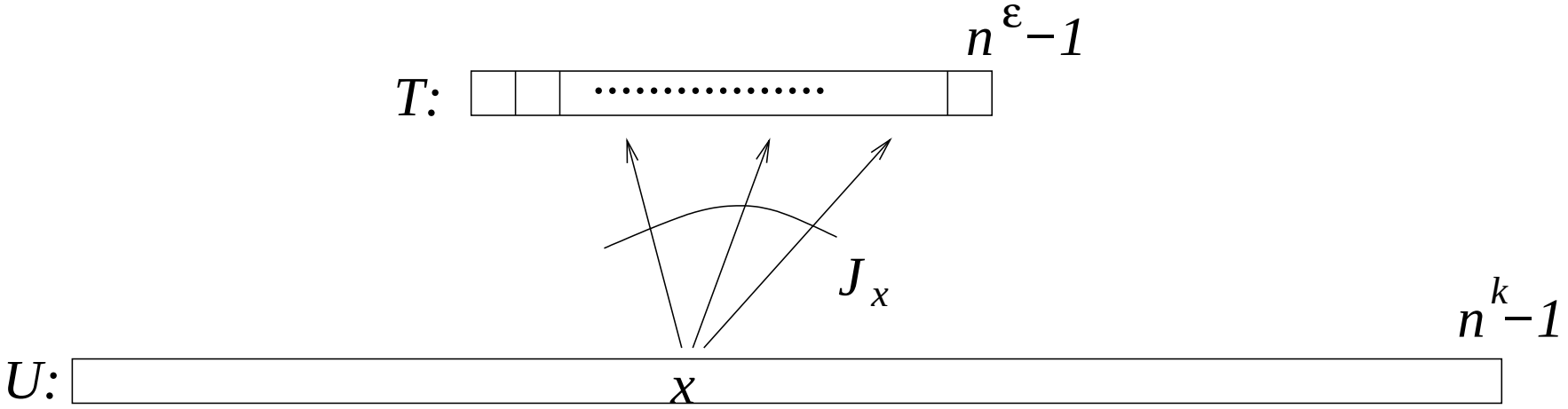
$$h(x) = \boldsymbol{T}[\boldsymbol{g}(x)] \oplus \boldsymbol{f}(x).$$

One table-lookup, "correction" by d-wise independent summand. Very efficient.

Analyzed in **[D./Meyer auf der Heide 1990]**, "class $\mathcal{R}$".

Many properties similar to fully random functions, notably distribution to hash values similar to the fully random case, **high reliability** features.

C) Siegel's functions, triangle property



C) Siegel's functions, triangle property

[Siegel 1989/2004] (second of two constructions)

$U = [n^k]$.

Table $\boldsymbol{T}[0..r-1]$, $r = n^\varepsilon$ random entries from $R = (R, \oplus)$.

Plus mapping $U \ni x \mapsto J_x \subseteq [r]$, with all J_x of fixed size k, **cleverly stored** ("folded weak expander graph").

$$h(x) = \bigoplus_{j \in J_x} T[j].$$

C) Siegel's functions, triangle property

	$[r]$							
	0	1	2	3	4	5	6	7
J_{x_1}:	0	1	0	0	1	0	0	1
J_{x_2}:	1	0	1	0	0	0	1	0
J_{x_3}:	0	1	0	0	1	0	1	0
J_{x_4}:	0	1	1	1	0	0	0	0
J_{x_5}:	1	0	1	0	0	1	0	0

C) Siegel's functions, triangle property

	$[r]$							
	7	5	0	1	2	3	4	6
J_{x_1}:	1	0	0	0	1	0	0	1
J_{x_5}:	0	1	1	0	0	1	0	0
J_{x_2}:	0	0	1	1	0	1	0	0
J_{x_3}:	0	0	0	1	1	0	0	1
J_{x_4}:	0	0	0	0	1	1	1	0

C) Siegel's functions, triangle property

Utilizes **triangle property**:

For all $X \subseteq U$, $|X| \leq n^\delta$, the $|X| \times r$ indicator matrix

$$(a_{xj})_{\substack{x \in X \\ j \in [r]}} \text{ with } a_{xj} = \begin{cases} 1 & \text{, if } j \in J_x, \\ 0 & \text{, if } j \notin J_x \end{cases}$$

can be brought into **echelon** form by row and column exchanges.

Lemma If n^δ-triangle property holds for the sets J_x, then the class of all functions $x \mapsto \bigoplus_{j \in J_x} \boldsymbol{T}[j]$ is n^δ-wise independent (hence $O(\log n)$-wise independent).

Evaluation time: $\boldsymbol{O(1)}$ (impractically high constant).

From 1990 till now Siegel's construction made it possible in many hashing applications to avoid the FRA.

D) d-wise Independence via Tabulation: Triangle Property Again

[Thorup/Zhang 2003+2010] use the same principle to arrive at **very efficient** d-universal classes for $d > 3$.

From $x \in U$ form a sequence of "derived characters" $\hat{x} = (\hat{x}_1, \ldots, \hat{x}_k) \in [u_1] \times \ldots \times [u_k]$ for $u_1, \ldots, u_k \leq n^\delta$.

$\hat{x}$ translates into a set $J_x \subseteq [u_1 + \ldots + u_k]$ of size k: (juxtapose $[u_1], \ldots, [u_k]$. — *E. g.*:

$\hat{x} = (2, 1, 3) \rightsquigarrow (0, 0, 1, 0; 0, 1, 0, 0; 0, 0, 0, 1)$
$\rightsquigarrow J_x = \{2, 6, 11\}$.

Map $x \mapsto J_x$ is required to have d-triangle property. (Called "unique character property" in **[Thorup/Zhang 2003+2010]**.)

D) d-wise Independence via Tabulation

k tables $\boldsymbol{T_1}[0..u_1 - 1], \ldots, \boldsymbol{T_k}[0..u_k - 1]$, total size $O(n^\delta)$, random entries from $(R, \oplus)$, and

$$h(x) = \bigoplus_{1 \le j \le k} T[\hat{x}_j].$$

Extremely efficient, if $(R, \oplus)$ is $\{0,1\}^\ell$ with XOR and the calculation of the derived characters is fast.

[Thorup/Zhang 2003+2010] use linear algebra (multiplication with a suitable generator matrix from coding theory) for the map $x \mapsto \hat{x}$ to ensure d-triangle property.

E) Linear Independence Implies Stochastic Independence

	$[r]$							
	0	1	2	3	4	5	6	7
J_{x_1}:	0	1	1	0	1	0	0	1
J_{x_2}:	1	0	1	1	0	1	0	0
J_{x_3}:	0	1	0	0	1	0	1	0
J_{x_4}:	0	0	1	0	1	1	0	1
J_{x_5}:	1	0	0	1	0	1	1	0

Triangle property not satisfied, but rank $= 5$.

E) Linear Independence Implies Stochastic Independence

If $T[0..r-1]$ contains **bit vectors** from $R = \{0,1\}^\ell$, with bitwise XOR as group operation $\oplus$, we can weaken the triangle condition:

Lemma (Folklore) Assume the mapping $x \mapsto J_x \subseteq [r]$ is such that for all sets $X \subseteq U$ of size d the indicator matrix

$$(a_{xj})_{\substack{x\in X\\ j\in[r]}} \quad \text{with} \quad a_{xj} = \begin{cases} 1 & \text{, if } j \in J_x, \\ 0 & \text{, if } j \notin J_x \end{cases}$$

has **linearly independent rows**.
Then $h(x) = \bigoplus_{j\in J_x} T[j]$ defines a d-wise independent class.

E) Linear Independence Implies Stochastic Independence

Proof: Elementary linear algebra: row rank = column rank.

From d-triangle property we get d-linear independence, hence d-wise stochastic independence of hash function.

Even more: $2d$-triangle property implies $(2d+1)$-wise linear independence over $\mathbb{Z}_2$, hence $(2d+1)$-wise (stochastic) independence. (Simplifies arguments in **[Thorup/Zhang 2010]**.)

Also observed in **[Klassen/Woelfel 2011]**.

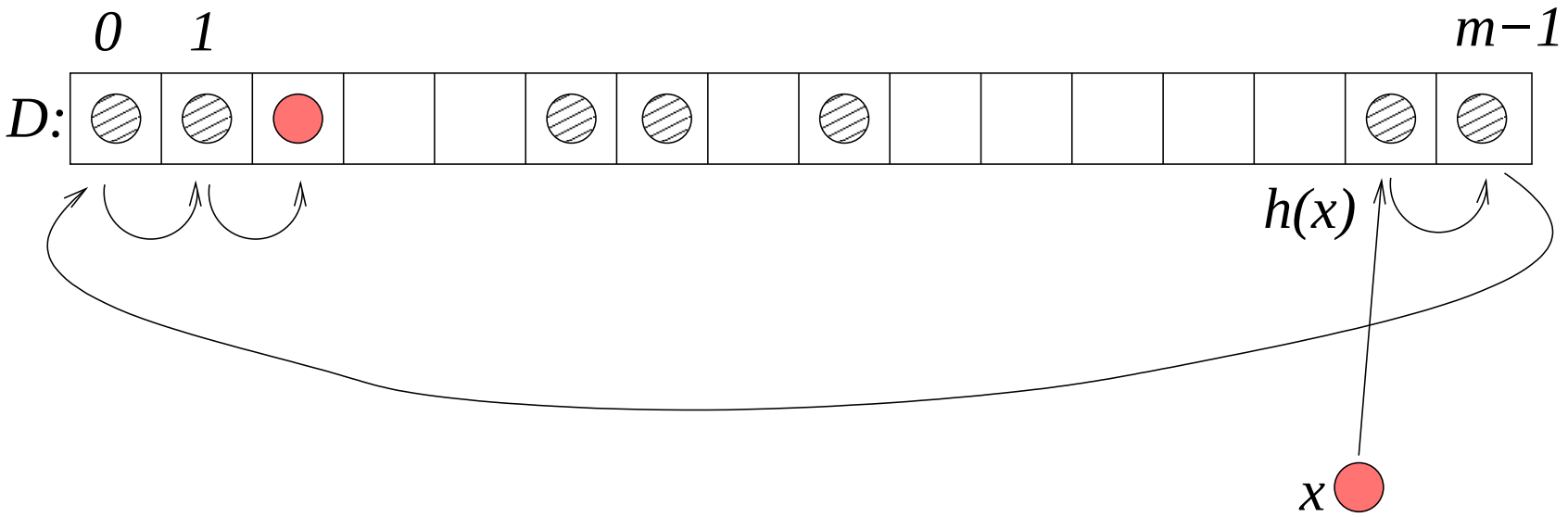
Part 2: The Scenes

Applications and How they React to Hash Classes

Applications: Algorithms/Constructions that use hash functions

- Linear probing
- Cuckoo hashing
- . . . with a stash
- Generalized cuckoo hashing
- Balanced allocation
- Simulation of uniform hashing
- Bloom filters
- Retrieval/Representation of functions

Application 1: Linear probing



Application 1: Linear probing

Set $S \subseteq U$, $S = \{x_1, \ldots, x_n\}$.

Implementation of the data type **Dictionary**:

Insert keys x from S into a table $D[0..m-1]$:

Check $D[h(x)]$, $D[h(x)+1]$, . . . , (indices modulo m)

until empty cell is found. Place x in this cell.

Searching: Same procedure, same running time.

[Knuth 1963] found that if $m = (1+\varepsilon)n$ the expected number of probes for inserting $y \notin S$ is $O(1/\varepsilon^2)$ (and much more), with FRA.

(There is a recent simple proof of $O(1)$ expected insertion time for the classroom, by Eppstein and Goodrich.)

Linear probing

[Schmidt/Siegel 1990]: $\Omega(\log n)$-wise independence is enough to guarantee essentially the same behavior as a fully random hash function.

Evaluation time?
$\Omega(\log n)$ with polynomials, $O(1)$ with Siegel's class.

Recent: 5-wise independence suffices!

Linear probing

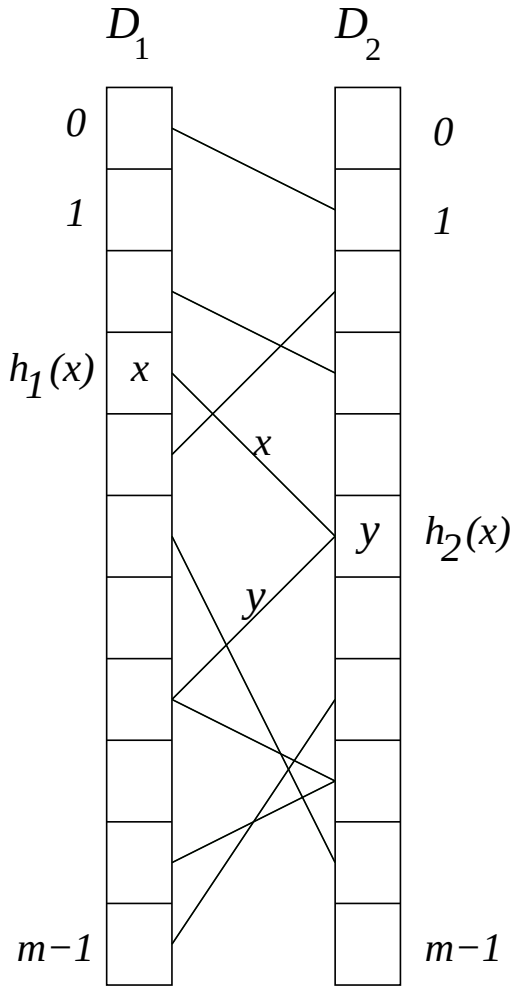
[Pagh/Pagh/Ružić 2007/09]: 5-wise independence implies $O(1/\varepsilon^{13/6})$ expected insertion time.
Contrast: Linear polynomials cause $\Omega(n \log n)$ total expected insertion time on some set S.

[Pătraşcu/Thorup 2010]: may have 4-wise independence but $\Omega(\log n)$ expected insertion time.
(A class of hash functions contrived for the effect.)

Theorem **[Pătraşcu/Thorup 2010]**
Linear probing with a hash function from the simple tabulation class has expected insertion time $\Theta(1/\varepsilon^2)$.

Ad-hoc proof, uses concentration bounds, avoids d-wise independence.

Application 2: Cuckoo Hashing



Application 2: Cuckoo Hashing

[Pagh/Rodler 2001/04]

Two hash tables $D_1[0..m-1], D_2[0..m-1]$, hash functions h_1, h_2.

Rules: Each table cell can hold one key.
$x \in S$ must be stored either in $D_1[h_1(x)]$ or in $D_2[h_2(x)]$.

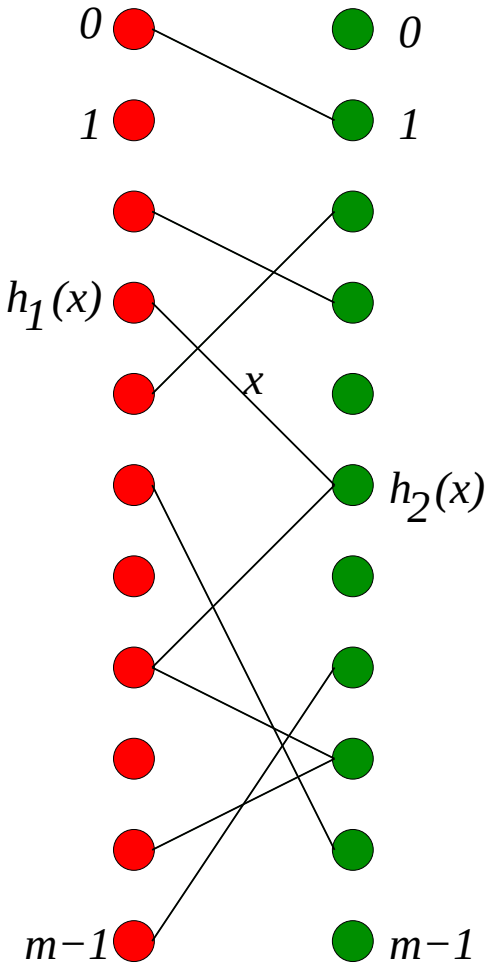
Results in guaranteed constant lookup time.

"Cuckoo hashing" because of an interesting insertion procedure (not important here).

Definition

h_1, h_2 are **suitable** for S if S can be stored with h_1, h_2 according to the two rules.

Cuckoo Hashing



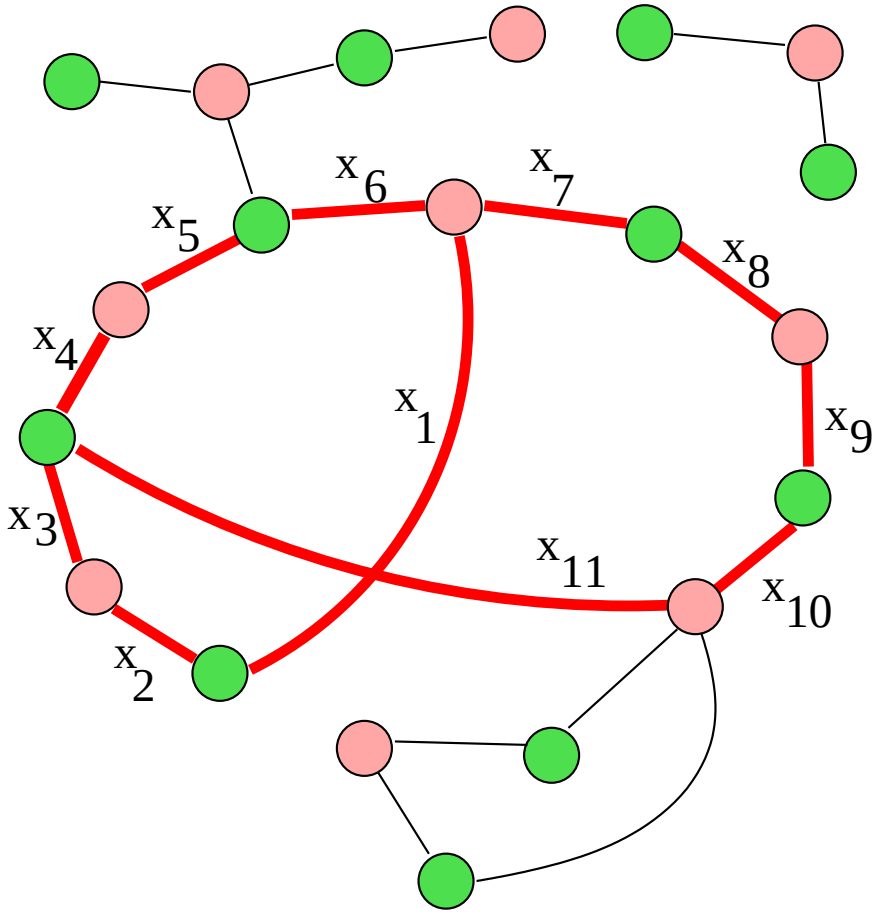
Essential:

The bipartite **cuckoo graph** $G(S, h_1, h_2)$.

Node set: two disjoint copies of $[m]$.

Edges: $(h_1(x), h_2(x))$, for $x \in S$.

Cuckoo Hashing



Fact: h_1, h_2 suitable for S $\Leftrightarrow$ connected components of $G(S, h_1, h_2)$ are trees or unicyclic.

"**Minimum obstructing subgraphs**": two connected cycles or one cycle with a chord.

Such a thing exists in $G(S, h_1, h_2)$ $\Leftrightarrow$ h_1, h_2 not suitable.

Cuckoo Hashing

Theorem **[Pagh/Rodler 2001/04]**

If $m \geq (1+\varepsilon)n$, and h_1, h_2 are $\Omega(\log n)$-wise independent, then h_1, h_2 are suitable for S with probability $1 - O(1/n)$.

Insertion works in expected constant time.

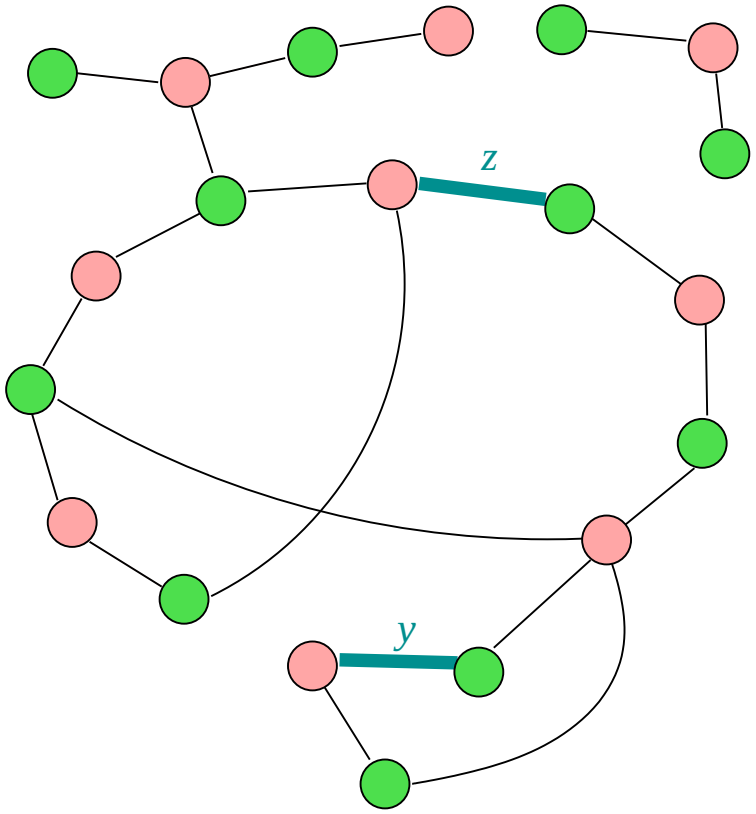
(Need Siegel's class.)

Theorem **[Pătraşcu/Thorup 2010]**

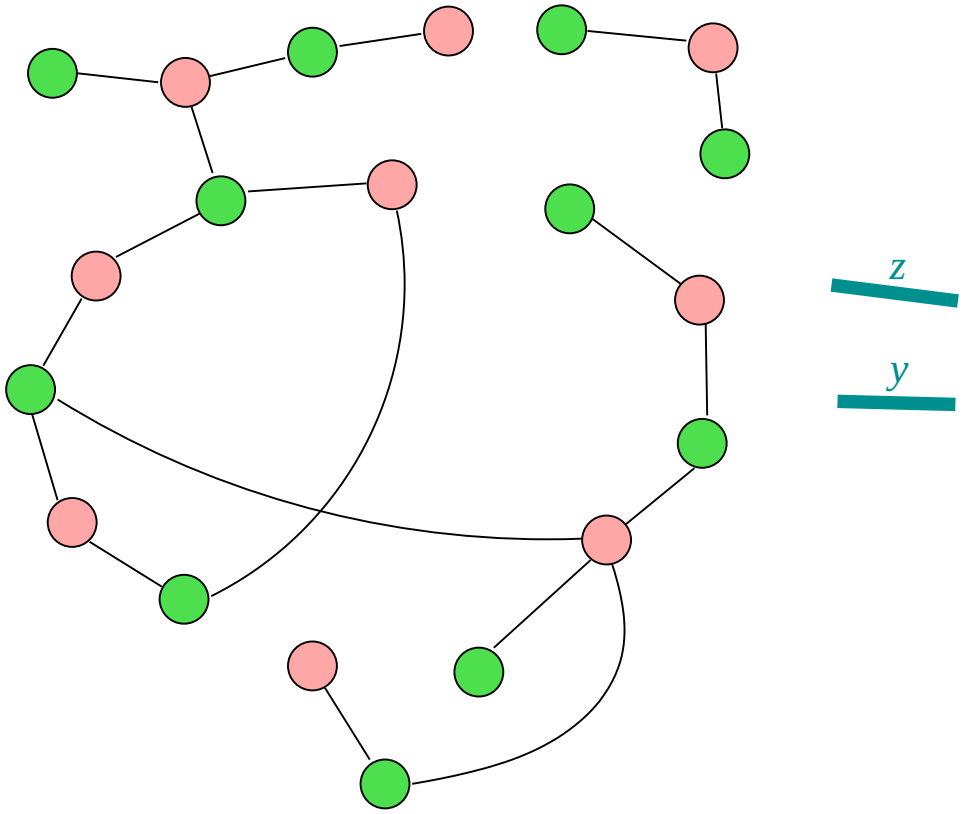
If $m \geq (1+\varepsilon)n$, and h_1, h_2 are from the tabulation class, then h_1, h_2 are suitable for S with probability $1 - O(1/n^{1/3})$.

Ad-hoc proof, avoids d-wise independence.

Cuckoo Hashing with "stash"



Cuckoo Hashing with "stash"



Cuckoo Hashing with "stash"

Objection raised by **[Kirsch/Mitzenmacher/Wieder 2008]**: failure probability $\Theta(1/n)$ may be too big.

Proposal: **"stash"**:
Put up to $s = O(1)$ keys into extra registers, to reduce the cuckoo graph to "tree or unicyclic component" shape.

Failure probability: $O(1/n^{s+1})$.

But: The analysis uses FRA (and some heavier machinery like Markov chain coupling).

Limitation of entropy argument

Cuckoo hashing provides another counterexample to using the Mitzenmacher/Vadhan argument (2-wise independence + entropy in keys = full randomness) indiscriminately.

Theorem **[D./Schellbach 2009]**
Let $U = [p]$ for p a prime number, and let $m = 2n \geq p^{6/7}$.
(Table is only one quarter full.)
Then for $h_i(x) = ((\boldsymbol{a_i} \cdot x + \boldsymbol{b_i}) \bmod p) \bmod m$, $i = 1, 2$, and $\boldsymbol{S} \subseteq U$ of size n **fully random**, h_1, h_2 are **not suitable** for S with high probability.

(And: Similar theorem even if two distinct primes are used; similar theorems for "multiplicative class".)

Limitation of entropy argument

Proof idea: The **full** cuckoo graph $G(\boldsymbol{U}, h_1, h_2)$ is very regular, for almost all pairs h_1, h_2.
W. h. p. it contains $\Omega(n)$ small disjoint copies of obstructing substructures like two cycles connected by a path.
Since S is very dense in U, the probability that one of these obstructing substructures is "switched on" by a random S is high.

Same effect occurs if keys are chosen randomly from certain small subsets of U ("structured keys").

Fortunately: Cuckoo hashing can be done safely with cheap hash functions . . .

Another hash class: Two tables plus three functions

Use three d-wise independent hash functions:
$\boldsymbol{f_1}, \boldsymbol{f_2}: U \to R$ and $\boldsymbol{g}: U \to [r]$ and random tables
$\boldsymbol{T_1}[0..r-1]$ and $\boldsymbol{T_2}[0..r-1]$ with entries from R. Define

$$h_1(x) := \boldsymbol{T_1}[g(x)] \oplus f_1(x) \quad \text{and} \quad h_2(x) := \boldsymbol{T_2}[g(x)] \oplus f_2(x)$$

Resulting class $\boldsymbol{\mathcal{R}_2}$ has hash function **pairs**.
Ad-hoc analysis in **[D./Woelfel 2003]**.

If $m \geq (1+\varepsilon)n$ and d is a sufficiently large constant, then $G(S, h_1, h_2)$ has many features of random graphs.

Sufficient to push through analysis of cuckoo hashing.

Ad-hoc analysis of cuckoo hashing

Let S be given. We have:

$$\begin{aligned}
&\Pr(h_1, h_2 \text{ not suitable for } S) \\
\leq &\Pr(\exists X \subseteq S\colon G(X, h_1, h_2) \text{ is minimal obstructing substructure}) \\
\leq &\sum_{X \subseteq S} \sum_{\substack{H\colon H \text{ has } |X| \text{ edges} \\ H \text{ is minimal obstructing}}} \Pr(G(X, h_1, h_2) \text{ “realizes” } H).
\end{aligned}$$

This evaluates to $O(1/n)$ for **fully random** hash functions.

For $\mathcal{R}_2$, we find there is a **small** exception set $B_S \subseteq \mathcal{R}_2$, with **(small)** probability $O(n/r^{d/2})$ that captures all “risks”:

- Outside B_S: $G(X, h_1, h_2)$ is fully random if it is connected.

Ad-hoc analysis of cuckoo hashing

Theorem **[D./Woelfel 2003]**

$\Pr(h_1, h_2 \text{ not suitable for } S) = O(1/n) + O(n/r^{d/2})$.

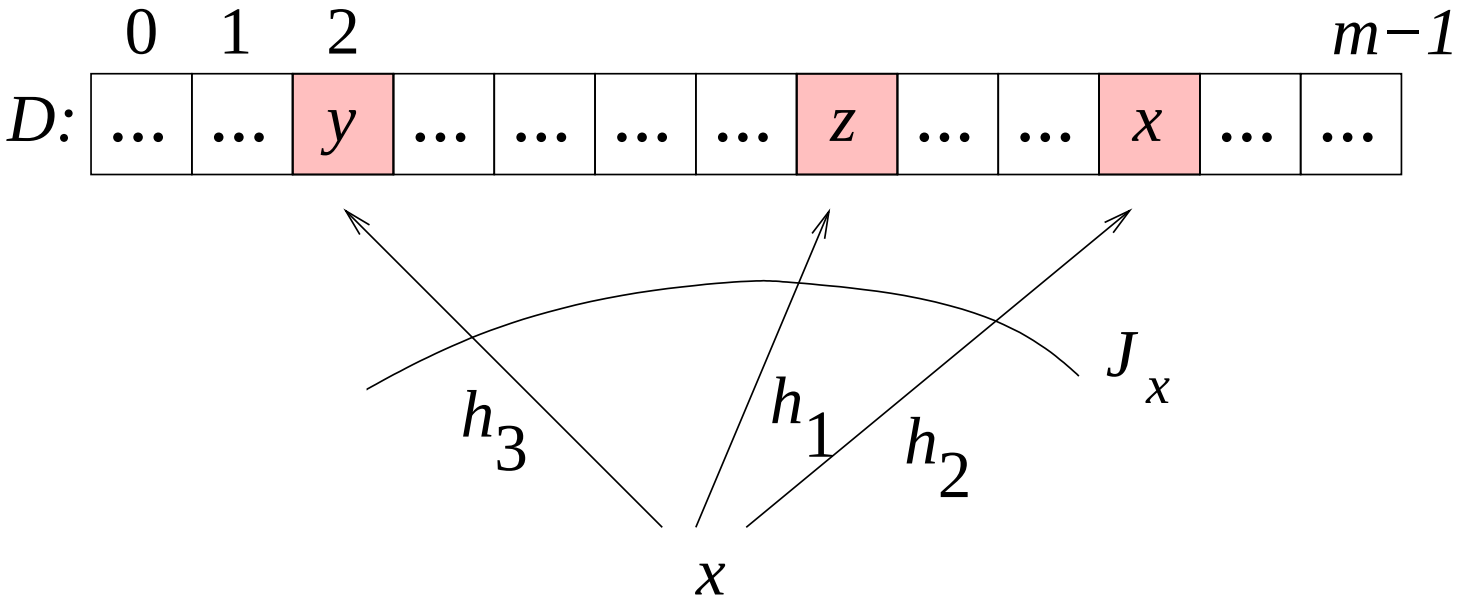
Can be generalized to cuckoo hashing **with a stash**.

Minimal obstructing substructure for stash size s: Cycles plus paths connecting cycles such that at least $s+1$ edges must be removed to get trees and unicyclic components.

[Aumüller 2010], **[Aumüller/D./Woelfel, forthcoming]**:
Can generalize analysis with $\mathcal{R}_2$ from standard cuckoo hashing to the situation with a stash. Yields:

$$\Pr(h_1, h_2 \text{ not suitable for } S, \text{ stash size } s) = O\left(\frac{1}{n^{s+1}}\right) + O\left(\frac{n}{r^{d/2}}\right).$$

Application 3: Generalized Cuckoo Hashing



Application 3: Generalized Cuckoo Hashing

[Fotakis,Pagh,Sanders,Spirakis 2004/05].

One hash table $D[0..m-1]$, k hash functions $h_1, \ldots, h_k$.

Not hard to arrange: $J_x = \{h_1(x), \ldots, h_k(x)\}$ has size k.

Rules: Each table cell can hold one key.
$x \in S$ must be stored either in $C[h_1(x)]$ or . . . or in $C[h_k(x)]$.

$h_1, \ldots, h_k$ are **suitable** for $S = \{x_1, \ldots, x_n\}$ if the keys can be placed according to these rules.

Generalized Cuckoo Hashing

Theorem (Thresholds for generalized c. h.)

For each $k \geq 3$ there is $c_k < 1$ (with $1 - c_k \sim e^{-k}$) such that: if $J_x = \{h_1(x), \ldots, h_k(x)\}$ are fully random on S, then:

(a) If $n/m = c < c_k$ and $n, m \to \infty$, then $h_1, \ldots, h_k$ are suitable for S w. h. p.

(b) If $n/m = c > c_k$ and $n, m \to \infty$, then $h_1, \ldots, h_k$ are not suitable for S w. h. p.

Proved independently (using FRA) by
[Frieze/Melsted 2009], **[Fountoulakis/Panagiotou 2009/10]**, **[D./Goerdt/Mitzenmacher/Montanari/Pagh/Rink 2009/10]**.

Generalized Cuckoo Hashing - Proof idea

Consider the random $n \times m$ indicator matrix (generated by the system $J_x = \{h_1(x), \ldots, h_k(x)\}$, $x \in S$):

$$(a_{xj})_{\substack{x \in S \\ j \in [r]}} \text{ with } a_{xj} = \begin{cases} 1 & \text{, if } j \in J_x, \\ 0 & \text{, if } j \notin J_x. \end{cases}$$

Consider density thresholds for two events.

(1) $h_1, \ldots, h_k$ are suitable for S in the sense of generalized cuckoo hashing.

(2) $(a_{xj})_{x \in S, j \in [r]}$ has linearly independent rows.

It turns out the questions are "threshold-equivalent".

(Equivalent formulation: edge orientation in random hypergraphs.)

Generalized Cuckoo Hashing - Random Indicator Matrix

	$[m]$							
	0	1	2	3	4	5	6	7
J_{x_1}:	0	0	1	0	0	0	1	1
J_{x_2}:	0	0	1	0	1	0	1	0
J_{x_3}:	1	1	0	0	1	0	0	0
J_{x_4}:	0	0	1	0	0	1	0	1
J_{x_5}:	1	1	0	0	0	1	0	0

$n = 5$, $k = 3$, full row rank (!)

Generalized Cuckoo Hashing - Proof idea

The following are "threshold-equivalent" for the density n/m:

(1) $h_1, \ldots, h_k$ are suitable for S in the sense of generalized cuckoo hashing.

(2) $(a_{xj})_{x \in S, j \in [r]}$ has linearly independent rows.

Easy direction: (2) $\Rightarrow$ (1) holds even deterministically. If the rows are linearly independent, the matrix has a square submatrix with a nonzero determinant. This gives a one-to-one mapping $\sigma\colon S \to [m]$ with $a_{x,\sigma(x)} = 1$, or $\sigma(x) \in J_x$:
a cuckoo placement!

(1) $\Rightarrow$ (2) holds only threshold-wise.
Much more difficult proof (for $k = 3$: **[Dubois/Mandler 2002]**).

Generalized Cuckoo Hashing - Random Indicator Matrix

	$[m]$							
	0	1	2	3	4	5	6	7
J_{x_1}:	0	0	1	0	0	0	1	1
J_{x_2}:	0	0	1	0	1	0	1	0
J_{x_3}:	1	1	0	0	1	0	0	0
J_{x_4}:	0	0	1	0	0	1	0	1
J_{x_5}:	1	1	0	0	0	1	0	0

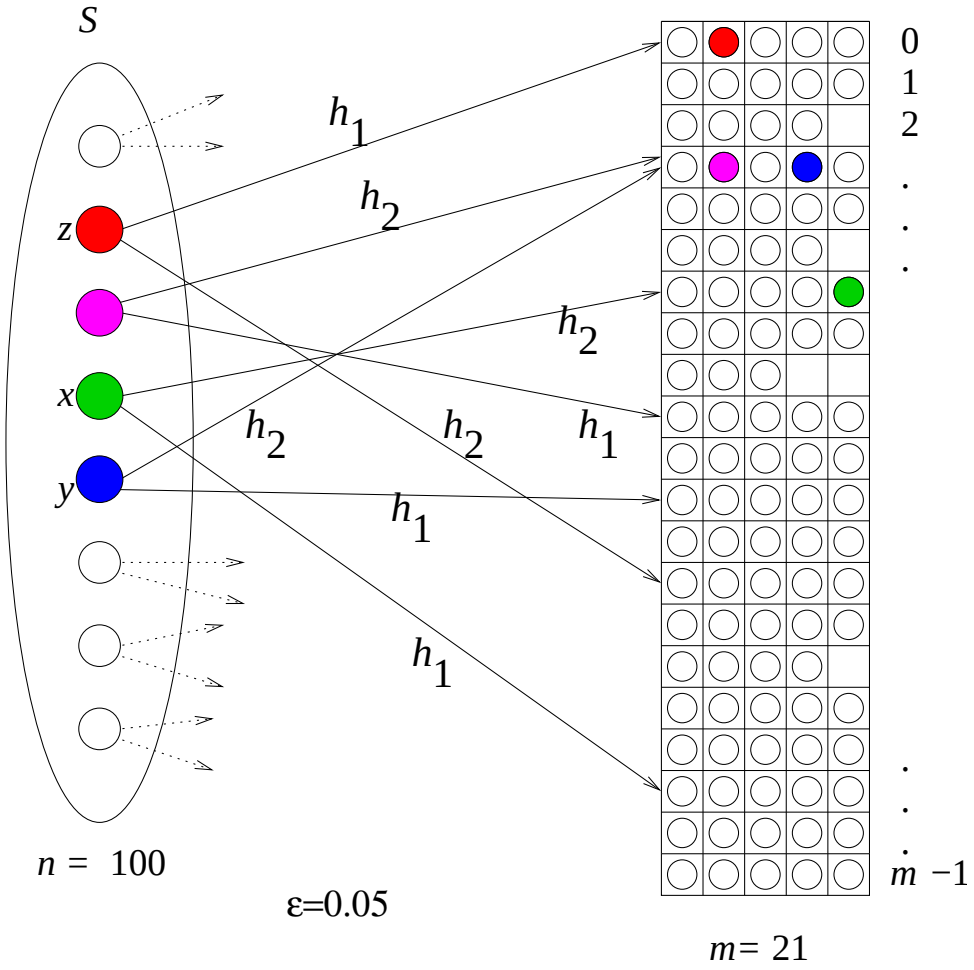
$n = 5$, $k = 3$, regular submatrix

Generalized Cuckoo Hashing - Random Indicator Matrix

	$[m]$							
	0	1	2	3	4	5	6	7
J_{x_1}:	0	0	1	0	0	0	1	1
J_{x_2}:	0	0	1	0	1	0	1	0
J_{x_3}:	1	1	0	0	1	0	0	0
J_{x_4}:	0	0	1	0	0	1	0	1
J_{x_5}:	1	1	0	0	0	1	0	0

$n = 5$, $k = 3$, regular submatrix $\rightarrow$ cuckoo placement.

Application 3: Generalized Cuckoo Hashing - Larger slots



Application 3: Generalized Cuckoo Hashing - Larger slots

[D./Weidling 2006], **[Cain/Sanders/Wormald 2007]**, **[Fernholz/Ramachandran 2007]**, **[Fountoulakis/Khosla/Panagioutou 2011]** consider generalizations with slots larger than 1.

These behave very nicely in experiments: dense packing, fast search because of locality.

The last three papers prove threshold results.

Analysis requires FRA.

Application 4: Simulating uniform hashing

. . . for sets $S \subseteq U$ of size n of means:

Given U and n, set up a **randomized** data structure **DS** that calculates a function $h_{\mathrm{DS}} : U \to R$ such that:

for each set $S \subseteq U$ of size n there is an event B_S ("bad") with $\Pr(B_S) = o(1)$ such that **on $\bar{B}_S$** the function h_{DS} behaves **fully randomly on S**.

Note: DS must work for arbitrary S.

The probabilities refer to the random components in DS.

Issues: Space for DS.
Evaluation time (always constant).

Simulating uniform hashing

[D./Meyer auf der Heide 1990] It is easy/trivial to simulate full randomness on sets S of size n, with space $n^{1+\varepsilon}$, e. g. $n^{3/2}$.

[D./Woelfel 2003], **[Pagh/Pagh 2003+08]**, independently:

(∗) Simulate full randomness on sets S of size n, with space $O(n)$.

[Pagh/Pagh 2008]

Generic transformation of (∗) to get space $(1+\varepsilon)n$, evaluation time $O(1/\varepsilon^2)$. Their basic construction involves Siegel's functions.

Simulating uniform hashing

[D./Woelfel 2003], **[Aumüller/D./Woelfel, forthcoming]**

Let $m = (1+\varepsilon)n$, set up
random tables $\boldsymbol{D_1}[0..m-1]$, $\boldsymbol{D_2}[0..m-1]$ with entries in R, and let

$$h(x) = \boldsymbol{D_1}[\boldsymbol{h_1}(x)] \oplus \boldsymbol{D_2}[\boldsymbol{h_2}(x)] \oplus \boldsymbol{h_3}(x),$$

where h_1, h_2 are as in cuckoo hashing and $\boldsymbol{(h_1, h_2, h_3)}$ jointly are in $\boldsymbol{\mathcal{R}'_3}$, with h_3 having range $\{0,1\}^\ell$.

This will simulate full randomness on sets of size n.

Total space: $2(1+\varepsilon)n$ words from R.

Efficient: it avoids Siegel's functions.

Simulating uniform hashing with "Split-and-Share"

Assume we had a randomized procedure that from $x \in U$ calculates a set $J_x \subseteq [m]$ of size k, such that:

for each $S \subseteq U$, w. h. p.: $(J_x)_{x \in S}$ is **fully random**.

For given n set up a data structure DS as follows.

Choose $\varepsilon_k > 2(1 - c_k) \sim 2e^{-k}$.

Let $m = (1 + \varepsilon_k)n$.

Fill table $\boldsymbol{T}[0..m-1]$ with random entries from $R = \{0,1\}^\ell$.

Define: $h(x) = \bigoplus_{j \in J_x} T[j]$.

Space: $(1+\Theta(e^{-k}))n$; evaluation time: $O(k) = O(\log(1/\varepsilon_k))$.

Simulating uniform hashing with "Split-and-Share"

Let S of size n be given. W. h. p. $(J_x)_{x \in S}$, is fully random.

Since n/m is well below c_k, w. h. p. the indicator matrix

$$(a_{xj})_{\substack{x \in S \\ j \in [r]}} \text{ with } a_{xj} = \begin{cases} 1 & \text{, if } j \in J_x, \\ 0 & \text{, if } j \notin J_x \end{cases}$$

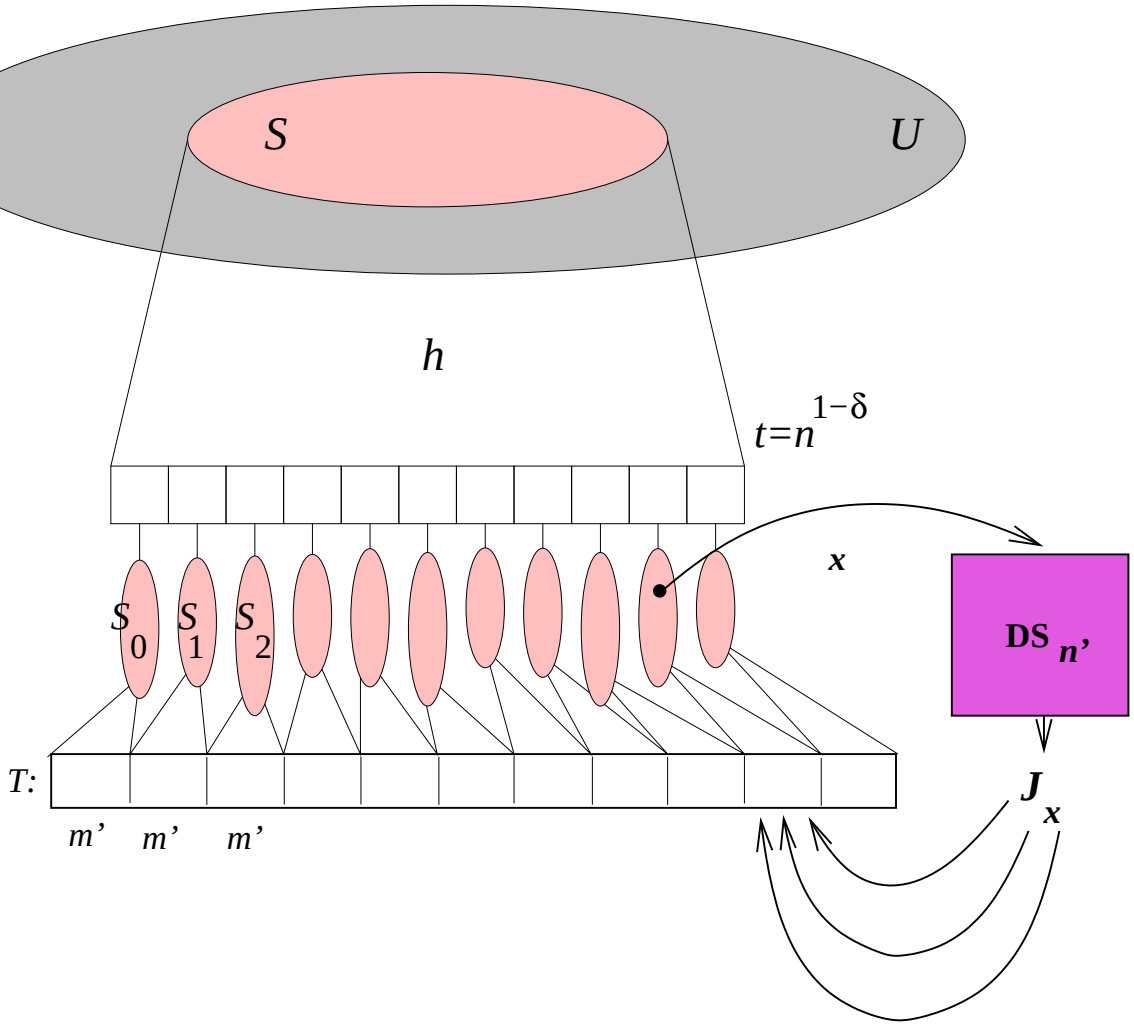
has full row rank; if so, h is fully random on S.

Nice joke! We're **using** sets J_x, **fully random** on S, to **construct** hash function h, **fully random** on S . . .

Go home, start over!

Solution in **[D./Rink 2009]**: **Split-and-Share**.

Trick: Split-and-Share



Trick: Split-and-Share

Arrange for a hash function g for splitting an arbitrary given set S into $t = n^{1-\delta}$ "chunks"

$$S_0, S_1, \ldots, S_{t-1}$$

of size $\leq n' = (1 + o(1))n^{\delta}$. (Easy, e. g. using class $\mathcal{R}$.)

Arrange **one** data structure $\mathbf{DS}_{n'}$ of size $O(n^{2\delta})$
(random numbers from $[m'] = [(1 + \varepsilon)n']$)
that maps a key x to a k-element subset J_x of $[m']$,
such that for any given S' of size n^{δ}
$(J_x)_{x \in S'}$ is fully random
(with a failure probability of $1/\text{poly}(n)$).

Trick: Split-and-Share

Set up $n^{1-\delta}$ random vectors $\boldsymbol{T_0}[0..m'-1], \ldots, \boldsymbol{T_{t-1}}[0..m'-1]$ with random entries from R.

For $x \in S$ the hash value is

$$h(x) = \bigoplus_{j \in J_x} T_{h(x)}[j].$$

Fully random (w. h. p.).

Space: $(1 + o(1))(1 + \varepsilon)n$ words.

Evaluation time: $O(\log(1/\varepsilon))$.

Asymptotically most compact simulation of uniform hashing known to date.

Split-and-Share: Other applications

Justify FRA for many applications in which it is possible to split the key set into chunks and treat the chunks separately.

- Siegel's construction: remove necessity for storing expander graphs
- hash tables with linear probing, double hashing, ideal hashing etc.
- Bloom filters and variants
- generalized Cuckoo hashing (more than 2 functions, larger bucket sizes)
- Balanced allocation **[Azar,Broder,Karlin,Upfal 1994/99]**, and following work.

Messages

- Fully random hash functions are a fruitful and helpful abstraction — but not a robust statement about behaviour of the "real world", not even in the Mitzenmacher/Vadhan version.
- FRA is a useful interface to modularize proofs. We use it in the analysis, take care of realizing this hypothesis some other time (but please before selling our product to the end consumer).
- to date: full randomness needed for dense random graphs, some versions of cuckoo hashing, Bloom filters, "hash-displace-compress", load balancing (dense case) and many others.

Messages

- Randomization with d-wise independence: big success story, clean math.
- Randomization with d-wise independence: limitations.
- Widely applicable justificaton of FRA: "Split-and-Share".
- Simulating full randomness: can be done close to the information theory space bounds, with efficient evaluation.
- in sparse graphs and hypergraphs: d-wise independence $+$ tables $=$ behaviour close to random graphs. Application: Load balancing, cuckoo hashing (with stash), simulation of full randomness.
- Current news: ad-hoc analysis of tabulation classes.

What lies ahead?

- Analyze more common schemes with ad-hoc analysis for suitable weaker hash classes.
  Tough: when the density n/m is high.
  - Bloom filtering
  - generalized cuckoo hashing
  - shared memory simulation
  - load balancing with higher loads
- Explore further applications of "Split-and-Share";
  reduce overhead.
- Explore power of tabulation further (simple and enhanced).

Thank you!