Strong Randomness Properties of (Hyper-)Graphs Generated by Simple Hash Functions

Martin Aumüller

Technische Universität Ilmenau, Germany

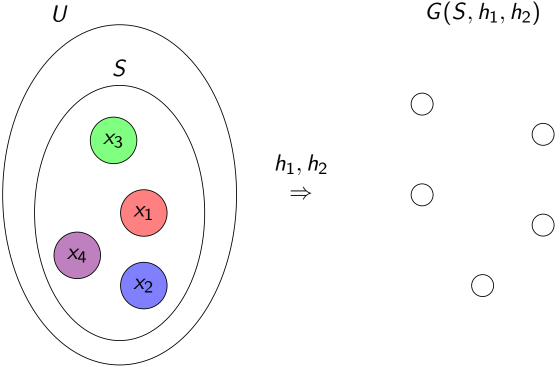
Mini-Workshop on Hashing
Summer School on Hashing 2014, Copenhagen

Joint work with: Martin Dietzfelbinger, Philipp Woelfel

Our Setting

- U totally ordered finite set, $S \subseteq U, S = \{x_1, x_2, \ldots, x_n\}$ s.t. $x_1 < x_2 < \cdots < x_n$. Let $m \in \mathbb{N}$
- $h_1, h_2 : U \to [m] = \{0, 1, \ldots, m-1\}$.

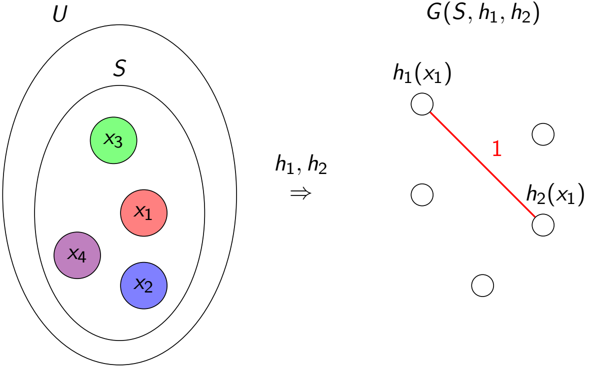
Build a labeled graph using S and h_1, h_2.



Our Setting

- U totally ordered finite set, $S \subseteq U, S = \{x_1, x_2, \ldots, x_n\}$ s.t. $x_1 < x_2 < \cdots < x_n$. Let $m \in \mathbb{N}$
- $h_1, h_2 : U \to [m] = \{0, 1, \ldots, m-1\}$.

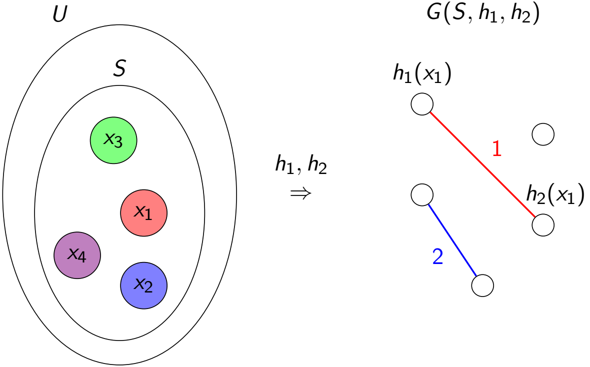
Build a labeled graph using S and h_1, h_2.



Our Setting

- U totally ordered finite set, $S \subseteq U, S = \{x_1, x_2, \ldots, x_n\}$ s.t. $x_1 < x_2 < \cdots < x_n$. Let $m \in \mathbb{N}$
- $h_1, h_2 : U \to [m] = \{0, 1, \ldots, m-1\}$.

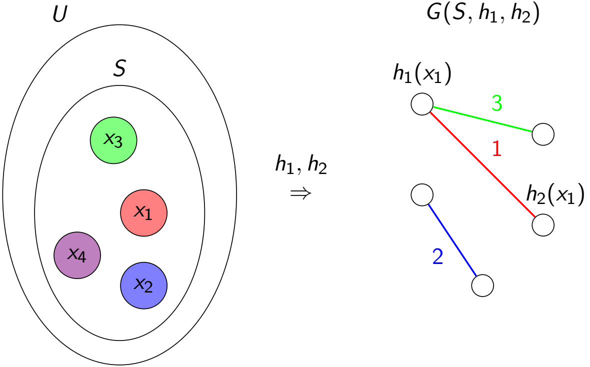
Build a labeled graph using S and h_1, h_2.



Our Setting

- U totally ordered finite set, $S \subseteq U, S = \{x_1, x_2, \ldots, x_n\}$ s.t. $x_1 < x_2 < \cdots < x_n$. Let $m \in \mathbb{N}$
- $h_1, h_2 : U \to [m] = \{0, 1, \ldots, m-1\}$.

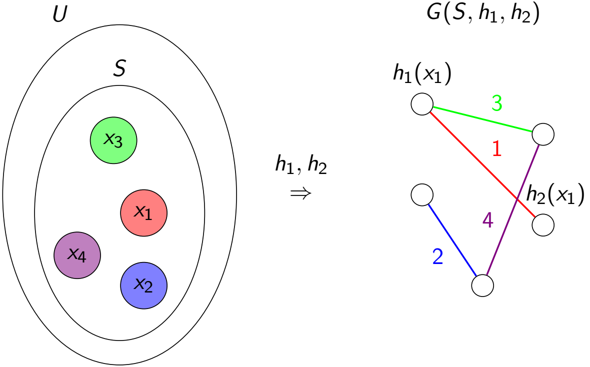
Build a labeled graph using S and h_1, h_2.



Our Setting

- U totally ordered finite set, $S \subseteq U, S = \{x_1, x_2, \ldots, x_n\}$ s.t. $x_1 < x_2 < \cdots < x_n$. Let $m \in \mathbb{N}$
- $h_1, h_2 : U \to [m] = \{0, 1, \ldots, m-1\}$.

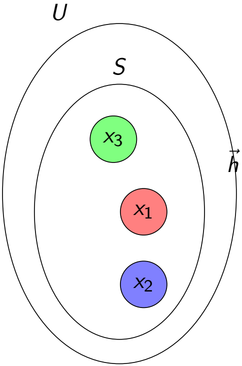
Build a labeled graph using S and h_1, h_2.



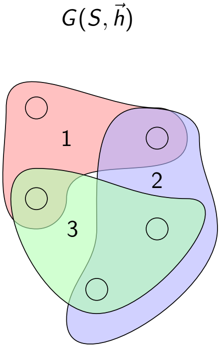
Our Setting

- U totally ordered finite set, $S \subseteq U, S = \{x_1, x_2, \ldots, x_n\}$ s.t. $x_1 < x_2 < \cdots < x_n$. Let $m \in \mathbb{N}$

Build a labeled d-uniform hypergraph using S and $\vec{h} = (h_1, \ldots, h_d)$.



$\vec{h} = (h_1, h_2, h_3)$
$\Rightarrow$



Part I

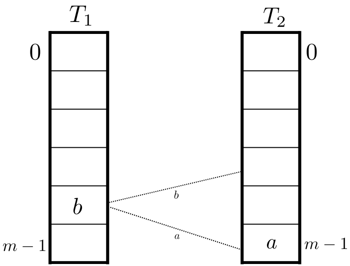
Random Graph Theory in the Analysis of Algorithms and Data Structures

Application 1: Cuckoo Hashing (Pagh/Rodler, 2001/2004)

A hashing-based implementation of the dictionary data type.

Setting:

- set $S \subseteq U$ of n keys
- two tables $T_1[0..m-1]$ and $T_2[0..m-1]$, $m \geq (1+\varepsilon)n$
- two (hash) functions h_1, h_2 with $h_i \colon U \to [m]$



Rules:

- each table cell can hold exactly one key
- a key x must be stored either in $T_1[h_1(x)]$ or $T_2[h_2(x)]$ (fast lookup and deletions!)

Definition

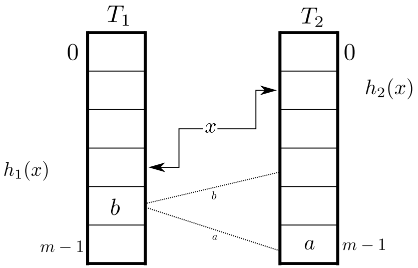
If S can be stored according to these rules, we call (h_1, h_2) **suitable for** S.

Application 1: Cuckoo Hashing (Pagh/Rodler, 2001/2004)

A hashing-based implementation of the dictionary data type.

Setting:

- set $S \subseteq U$ of n keys
- two tables $T_1[0..m-1]$ and $T_2[0..m-1]$, $m \geq (1+\varepsilon)n$
- two (hash) functions h_1, h_2 with $h_i\colon U \to [m]$



Rules:

- each table cell can hold exactly one key
- a key x must be stored either in $T_1[h_1(x)]$ or $T_2[h_2(x)]$ (fast lookup and deletions!)

Definition

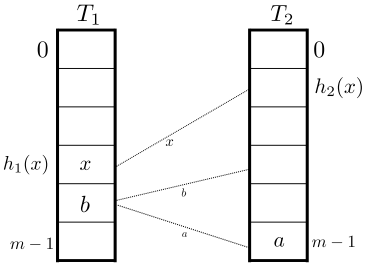
If S can be stored according to these rules, we call (h_1, h_2) **suitable for** S.

Application 1: Cuckoo Hashing (Pagh/Rodler, 2001/2004)

A hashing-based implementation of the dictionary data type.

Setting:

- set $S \subseteq U$ of n keys
- two tables $T_1[0..m-1]$ and $T_2[0..m-1]$, $m \geq (1+\varepsilon)n$
- two (hash) functions h_1, h_2 with $h_i \colon U \to [m]$



Rules:

- each table cell can hold exactly one key
- a key x must be stored either in $T_1[h_1(x)]$ or $T_2[h_2(x)]$ (fast lookup and deletions!)

Definition

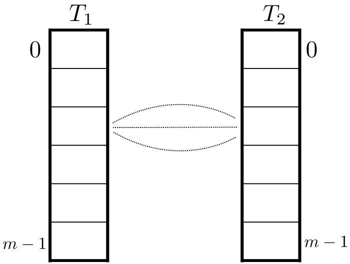
If S can be stored according to these rules, we call (h_1, h_2) **suitable for** S.

Application 1: Cuckoo Hashing (Pagh/Rodler, 2001/2004)

A hashing-based implementation of the dictionary data type.

Setting:

- set $S \subseteq U$ of n keys
- two tables $T_1[0..m-1]$ and $T_2[0..m-1]$, $m \geq (1+\varepsilon)n$
- two (hash) functions h_1, h_2 with $h_i\colon U \to [m]$



Rules:

- each table cell can hold exactly one key
- a key x must be stored either in $T_1[h_1(x)]$ or $T_2[h_2(x)]$ (fast lookup and deletions!)

Definition

If S can be stored according to these rules, we call (h_1, h_2) **suitable for** S.

Cuckoo Hashing: Failure Probability

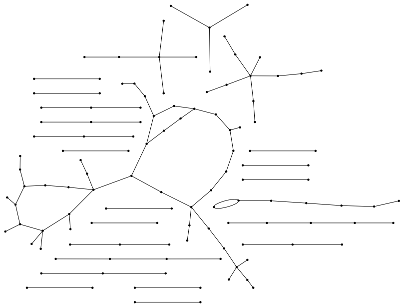
Theorem (Pagh/Rodler, 2004)

Let $S \subseteq U$ with $|S| = n$. If (h_1, h_2) are fully random (or $\Omega(\log n)$-wise independent), then

$$\Pr((h_1, h_2) \text{ unsuitable for } S) = O(1/n).$$

In fact: $\Theta(1/n)$.

The Cuckoo Graph - Example II



Lemma (Devroye, Morin 2003)

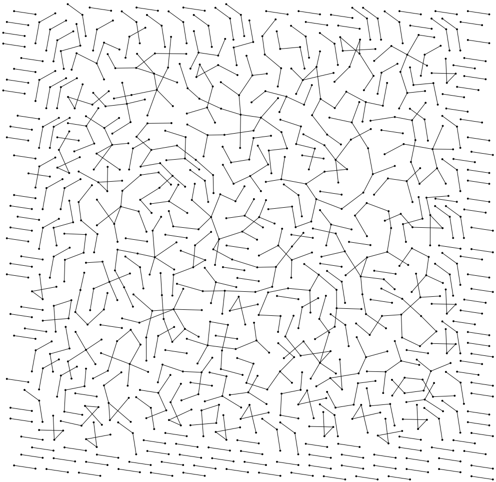
(h_1, h_2) suitable for S if and only if each connected component of $G(S, h_1, h_2)$ is either a tree or unicyclic.

Central Question: Does $G(S, h_1, h_2)$ contain a connected component with more than one cycle?

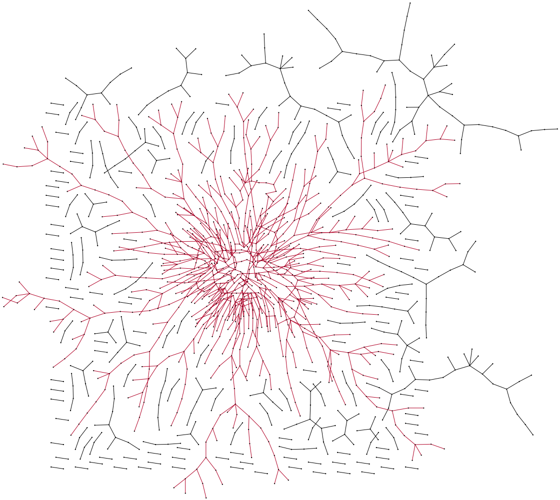
Fully Random Case (Erdős–Rényi, 1960)

If $m \geq (1 + \varepsilon)n$ then all connected components of $G(S, h_1, h_2)$ are trees or unicyclic with probability $1 - O(1/n)$.

Random graphs with $2(1+\varepsilon)n$ vertices and n edges look like this:



Random graphs with $2(1-\varepsilon)n$ vertices and n edges look like this:



Cuckoo Hashing with a Stash

(Kirsch, Mitzenmacher, Wieder, 2008)

- Failure probability: $\Theta(1/n)$ is too large.
- Proposal: Can put up to $s = O(1)$ keys into additional storage, called "stash"

Theorem (K/M/W, 2008)

Let $S \subseteq U$ with $|S| = n$. If (h_1, h_2) are **fully random**, then

$$\Pr((h_1, h_2) \text{ unsuitable for } S \text{ with stash size } s) = O(1/n^{s+1}).$$

- "Full Randomness Assumption" (FRA) central in their analysis
- constructions for FRA known, but undesirable

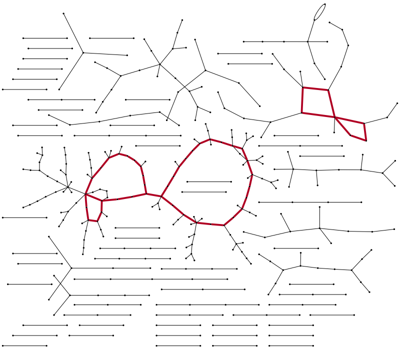
Cuckoo Hashing with a Stash/2

Excess: The minimal number of edges we have to remove from a graph such that each component contains at most one cycle.

Observation

(h_1, h_2) is unsuitable for S with stash size $s \Rightarrow G(S, h_1, h_2)$ contains a leafless subgraph with excess exactly $s + 1$.

Question: Does $G(S, h_1, h_2)$ contain a subgraph with excess $\geq s + 1$?



More Generalizations

- Deamortized Cuckoo Hashing (Arbitman, Naor, Segev, 2009)
  Question: How large are the connected components of $\leq \log n$ distinct keys w.h.p.?
- Generalized Cuckoo Hashing:
  - use $d \geq 3$ hash functions (Fotakis, Pagh, Sanders, Spirakis, 2003)
  - each table cell can hold $\ell \geq 2$ keys (Dietzfelbinger, Weidling, 2005)

  Questions on the orientability of (hyper-)graphs.
- Wear-minimization for Cuckoo Hashing (Eppstein, Goodrich, Mitzenmacher, Pszona, 2014)

Application 2: Randomized (Parallel) Load Balancing

(Stemann, 96)

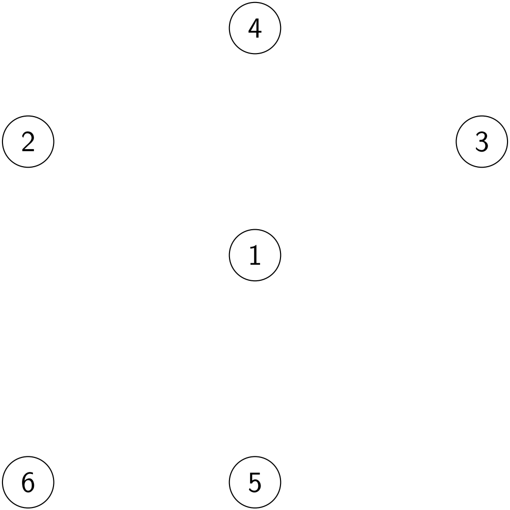
Set of jobs J, set of units U. $|J| = |U| := n$.
Each job chooses two candidate units. Task: Allocate jobs to units minimizing some goal (e.g., minimize maximum load on a unit).

Algorithm: c-collision protocol

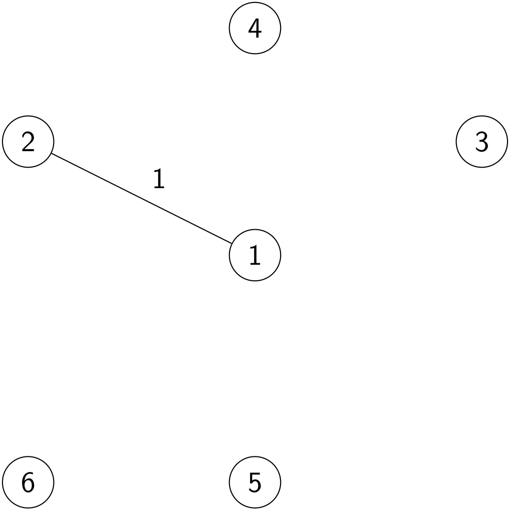
Each job chooses 2 units at random, then synchronously in rounds:

- Each unallocated job sends a request to its candidate units.
- If a unit gets at most c requests, it sends acknowledgements to all these jobs.
- Allocated jobs and units become inactive, next round starts.

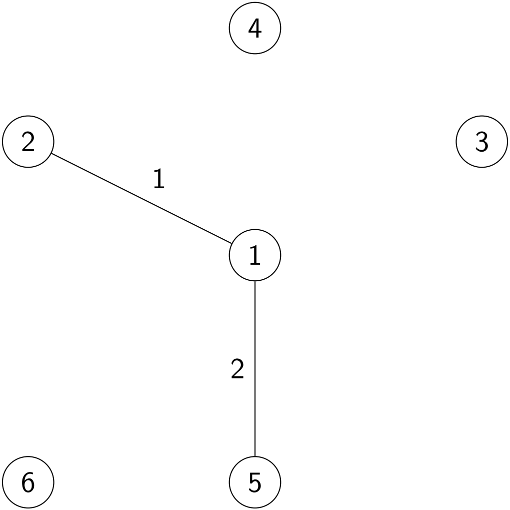
Example 2-collision protocol



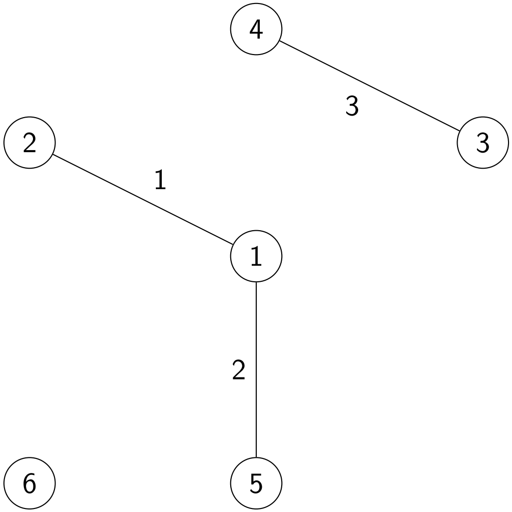
Example 2-collision protocol



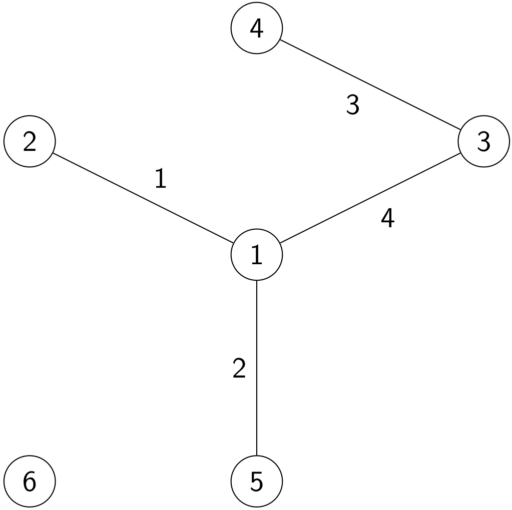
Example 2-collision protocol



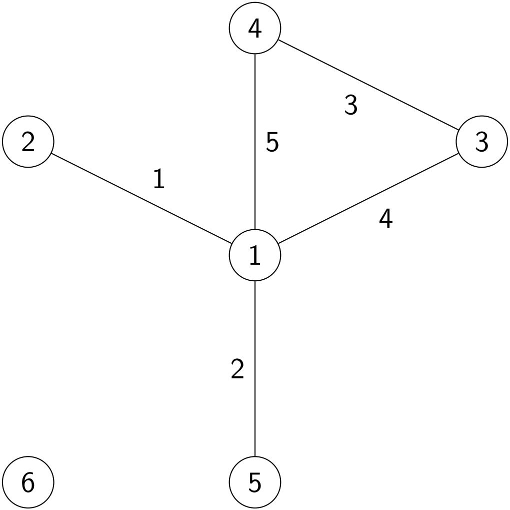
Example 2-collision protocol



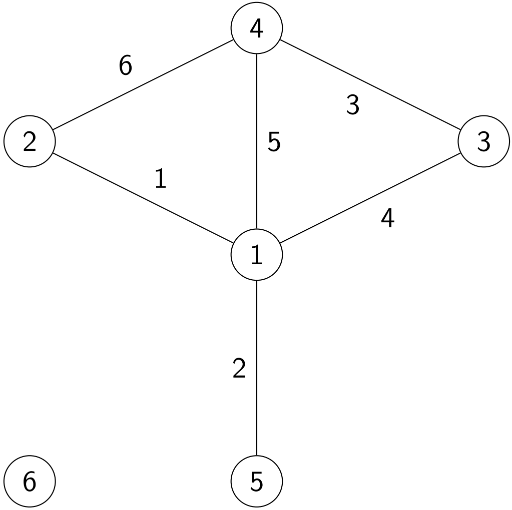
Example 2-collision protocol



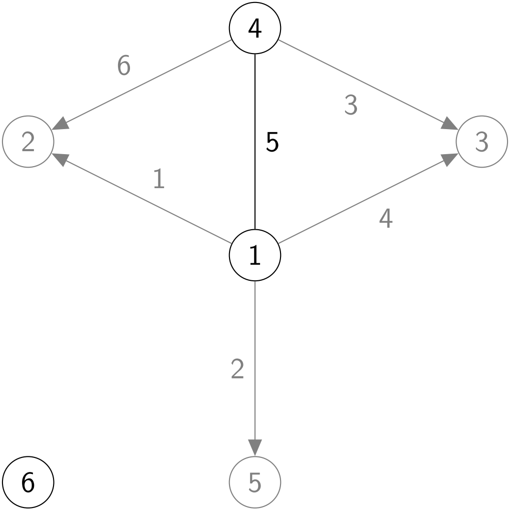
Example 2-collision protocol



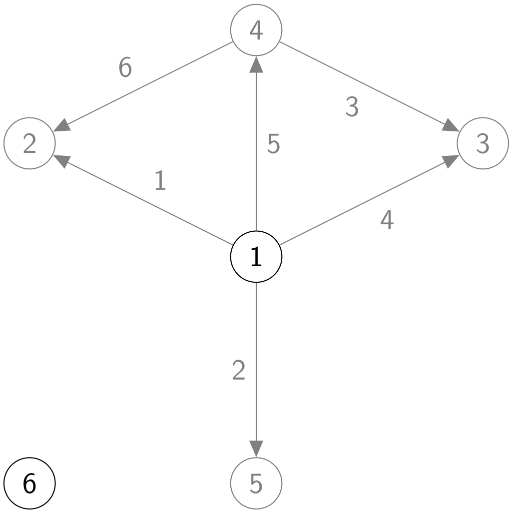
Example 2-collision protocol



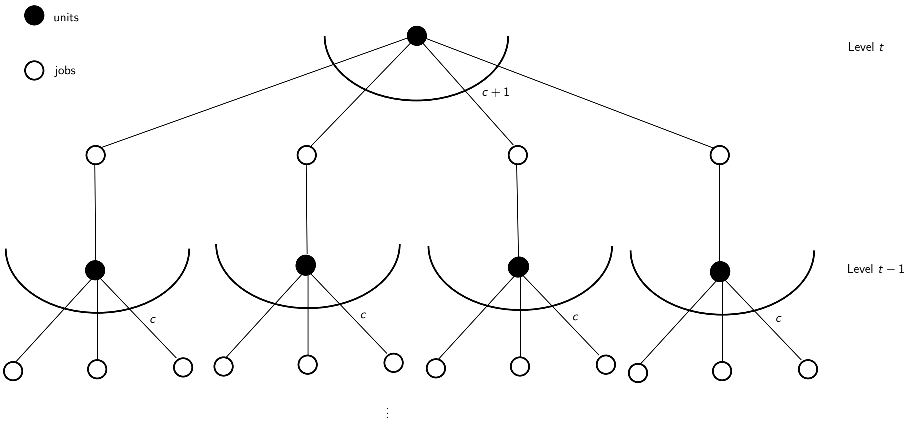
Example 2-collision protocol



Example 2-collision protocol



Randomized Load Balancing /2



Does $G(S, h_1, h_2)$ contain a **witness tree**?

First Moment Method

For the analysis to succeed, we have to prove that certain subgraphs do not occur in $G(S, h_1, h_2)$. Let A be the family of all such subgraphs. Let N^{A} denote the number of subgraphs in $G(S, h_1, h_2)$ which are in A.

Then:

$$\Pr\left(N^{\mathsf{A}} > 0\right) \leq \mathrm{E}\left(N^{\mathsf{A}}\right) = \sum_{G \in \mathsf{A}} \Pr\left(G \text{ is a subgraph of } G(S, h_1, h_2)\right).$$

Hash functions fully random: Analysis well understood.

This talk: Show how to apply this method for “simple hash functions”.

Related Work

- Many of the original paper show that $O(\log n)$-wise independence suffices. (Considered graphs have $O(\log n)$ edges.)
- Dietzfelbinger, Woelfel (2003): Class of hash functions with constant evalution time which (provably) allow running cuckoo hashing.
- Thorup, Patrascu (2011): Simple Tabulation Hashing allows running cuckoo hashing with slightly worse failure bounds than in the fully random case.
- Reingold, Rothblum, Wieder (2014): Cuckoo hashing (with a stash) and power of two choices with hash functions which have $O(\log n \log\log n)$ description length and $O((\log\log n)^2)$ evaluation time.

Part II

Graphs Generated by Simple Hash Functions

Key ingredients: linear polynomials & multiplication-shift

linear polynomials:

$$h_{a,b}(x) = ((a \cdot x + b) \mod p) \mod m,$$

where

- $p \geq |U|$ is a prime, and
- a and b are chosen uniformly at random from $\{0, \ldots, p-1\}$.

Key ingredients: linear polynomials & multiplication-shift

linear polynomials:

$$h_{a,b}(x) = ((a \cdot x + b) \mod p) \mod m,$$

where

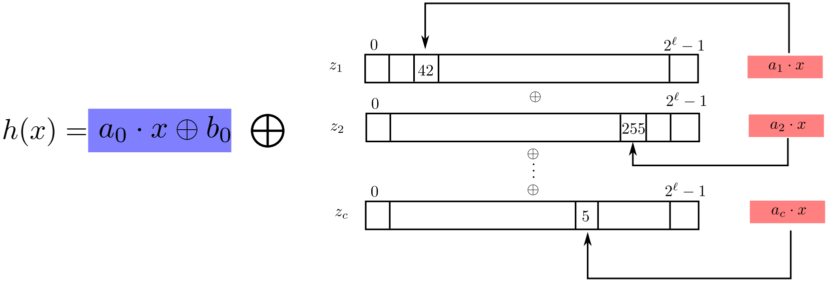
- $p \geq |U|$ is a prime, and
- a and b are chosen uniformly at random from $\{0, \ldots, p-1\}$.

"multiplication-shift":
From (Dietzfelbinger et al., 1997), for odd $a \in \{1, \ldots, 2^{32}\}$ (in 32-bit arithmetic):

$$h_a(x) = (ax) \gg (32 - \ell_{\mathrm{out}}).$$

Our hash class

For given $n \geq 1$, we combine linear polynomials & multiplication-shift with lookups in tables of size $2^\ell \approx \sqrt{n}$ filled with random values.



$$h_i(x) = f_i(x) \oplus \bigoplus_{j=1}^{c} z_j^{(i)}[g_j(x)], \qquad i = 1, 2$$

Class of all these pairs (h_1, h_2) **of hash functions**: $\mathcal{Z}$.

Extension of Hash class considered in (Dietzfelbinger, Woelfel, 2003)

Behavior of our hash class on fixed $T \subseteq S$

$$h_i(x) = \boxed{f_i(x)} \oplus \bigoplus_{j=1}^{c} z_j^{(i)}[\boxed{g_j(x)}], \qquad i = 1, 2$$

Central Observation

Let $T \subseteq S$. If there is a g_j such that at most one pair of keys in T collides under g_j (i.e., $g_j(x) = g_j(y)$), then h_1, h_2 are fully random on T.

- if this is the case: (h_1, h_2) T-**good**.
- otherwise (each g_j has more than one colliding pair of keys): (h_1, h_2) is T-**bad**.

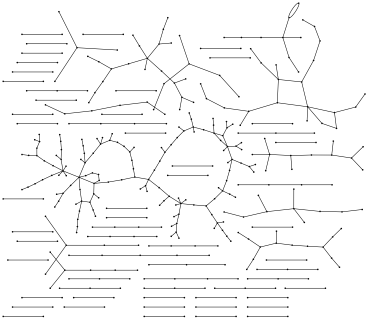
Example: Cuckoo Hashing with a Stash

Main Objective

For given S and stash size s, calculate

$$\Pr_{(h_1,h_2)\in\mathcal{Z}}((h_1, h_2) \text{ do not allow to store } S \text{ with stash size } s).$$

Recall: stash size s not sufficient $\Rightarrow$ there exists a subgraph s.t. one cannot remove s edges to obtain only tree or unicyclic components.



Minimal such "bad subgraph": a MOS_s. (Example: $s = 2$.)

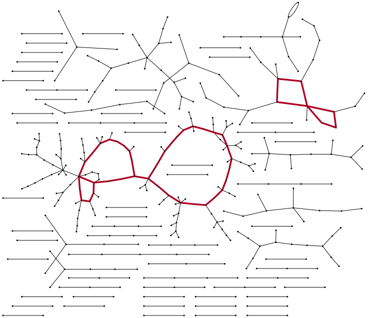
Example: Cuckoo Hashing with a Stash

Main Objective

For given S and stash size s, calculate

$$\Pr_{(h_1,h_2)\in\mathcal{Z}}((h_1,h_2) \text{ do not allow to store } S \text{ with stash size } s).$$

Recall: stash size s not sufficient $\Rightarrow$ there exists a subgraph s.t. one cannot remove s edges to obtain only tree or unicyclic components.



Minimal such "bad subgraph": a MOS_s. (Example: $s = 2$.)

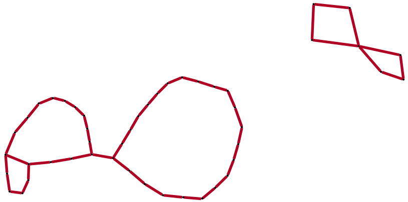
Example: Cuckoo Hashing with a Stash

Main Objective

For given S and stash size s, calculate

$$\Pr_{(h_1,h_2)\in\mathcal{Z}}((h_1,h_2) \text{ do not allow to store } S \text{ with stash size } s).$$

Recall: stash size s not sufficient $\Rightarrow$ there exists a subgraph s.t. one cannot remove s edges to obtain only tree or unicyclic components.



Minimal such "bad subgraph": a MOS_s. (Example: $s = 2$.)

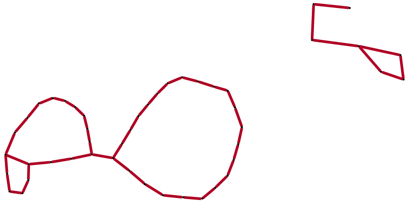
Example: Cuckoo Hashing with a Stash

Main Objective

For given S and stash size s, calculate

$$\Pr_{(h_1,h_2)\in\mathcal{Z}}((h_1, h_2) \text{ do not allow to store } S \text{ with stash size } s).$$

Recall: stash size s not sufficient $\Rightarrow$ there exists a subgraph s.t. one cannot remove s edges to obtain only tree or unicyclic components.



Minimal such "bad subgraph": a MOS_s. (Example: $s = 2$.)

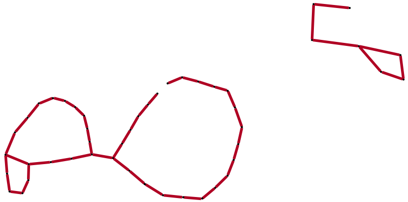
Example: Cuckoo Hashing with a Stash

Main Objective

For given S and stash size s, calculate

$$\Pr_{(h_1,h_2)\in\mathcal{Z}}((h_1, h_2) \text{ do not allow to store } S \text{ with stash size } s).$$

Recall: stash size s not sufficient $\Rightarrow$ there exists a subgraph s.t. one cannot remove s edges to obtain only tree or unicyclic components.



Minimal such “bad subgraph”: a MOS_s. (Example: $s = 2$.)

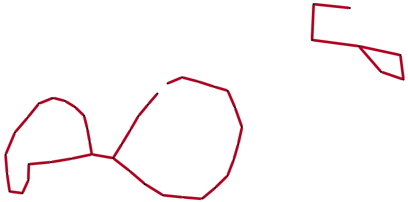
Example: Cuckoo Hashing with a Stash

Main Objective

For given S and stash size s, calculate

$$\Pr_{(h_1,h_2)\in\mathcal{Z}}((h_1,h_2) \text{ do not allow to store } S \text{ with stash size } s).$$

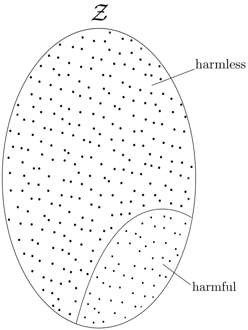
Recall: stash size s not sufficient $\Rightarrow$ there exists a subgraph s.t. one cannot remove s edges to obtain only tree or unicyclic components.



Minimal such “bad subgraph”: a MOS_s. (Example: $s = 2$.)

Collecting bad hash functions

We split our set of hash functions into "harmful" and "harmless" ones.



(h_1, h_2) are harmful, if there exists $T \subseteq S$ s.t.

- $G(T, h_1, h_2)$ forms a MOS_s, and
- (h_1, h_2) is T-bad.

$B^{\mathbf{MOS}_s}$:= the set of all the harmful pairs (h_1, h_2). (An event in our probability space!)

Splitting the calculation

We calculate:

$$\Pr(N_S^{\mathsf{MOS}_s} > 0) \;\leq\; \Pr(N_S^{\mathsf{MOS}_s} > 0 \cap \neg B^{\mathsf{MOS}_s}) \;+\; \Pr(B^{\mathsf{MOS}_s})$$

Splitting the calculation

We calculate:

$$\Pr(N_S^{\mathsf{MOS}_s} > 0) \ \leq \ \boxed{\Pr(N_S^{\mathsf{MOS}_s} > 0 \cap \neg B^{\mathsf{MOS}_s})} \ + \ \Pr(B^{\mathsf{MOS}_s})$$

- for **this** summand, we have

$$\Pr(N_S^{\mathsf{MOS}_s} > 0 \cap \neg B^{\mathsf{MOS}_s}) \leq \mathrm{E}^* \left(N_S^{\mathsf{MOS}_s} \right),$$

which is $O(1/n^{s+1})$.

Splitting the calculation

We calculate:

$$\Pr(N_S^{\mathsf{MOS}_s} > 0) \;\leq\; \boxed{\mathrm{E}^*\left(N_S^{\mathsf{MOS}_s}\right)} \;+\; \Pr(B^{\mathsf{MOS}_s})$$

- for **this** summand, we have

$$\Pr(N_S^{\mathsf{MOS}_s} > 0 \cap \neg B^{\mathsf{MOS}_s}) \leq \mathrm{E}^*\left(N_S^{\mathsf{MOS}_s}\right),$$

which is $O(1/n^{s+1})$.

(Such subgraphs have $|E| = |V| + s + 1$ edges, and there are only $|E|^{O(s)}$ such graphs (unlabeled).)

$$\sum_{t=s+2}^{n} \frac{n^t \cdot 2^s \cdot m^{t-s-1} \cdot t^{O(s)}}{m^{2t}} = \frac{1}{n^{s+1}} \sum_{t=s+2}^{n} \frac{t^{O(s)}}{(1+\varepsilon)^t} = O\left(\frac{1}{n^{s+1}}\right).$$

Splitting the calculation

We calculate:

$$\Pr(N_S^{\mathsf{MOS}_s} > 0) \; \leq \; \boxed{\mathrm{E}^*\left(N_S^{\mathsf{MOS}_s}\right)} + \boxed{\Pr(B^{\mathsf{MOS}_s})}$$

- for **this** summand, we have

$$\Pr(N_S^{\mathsf{MOS}_s} > 0 \cap \neg B^{\mathsf{MOS}_s}) \leq \mathrm{E}^*\left(N_S^{\mathsf{MOS}_s}\right),$$

which is $O(1/n^{s+1})$.

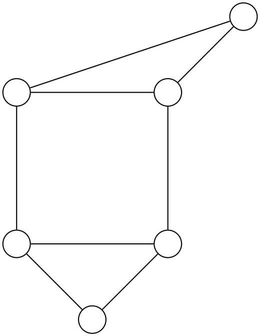
- **The hard part**: Calculating/bounding

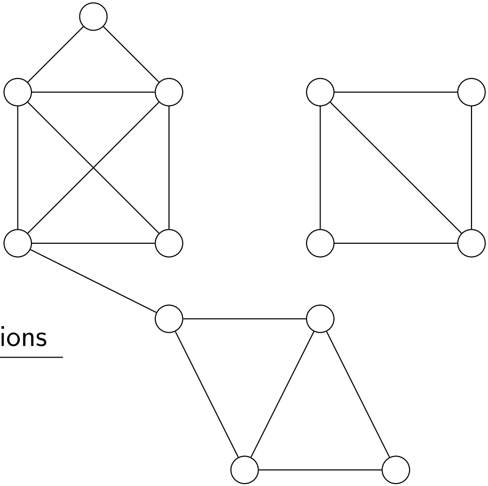
  $\Pr(B^{\mathsf{MOS}_s}) = \Pr(\exists T \subseteq S : G(T, h_1, h_2)$ forms a MOS_s $\cap (h_1, h_2)$ are T-bad $)$

- Wish: Use full randomness nonetheless
- Idea: Find a suitable event that contains B^{MOS_s}

Peeling of bad graphs (simplified)

Assume "$\exists T \subseteq S : G(T, h_1, h_2)$ forms a $\text{MOS}_s \cap (h_1, h_2)$ are T-bad ".

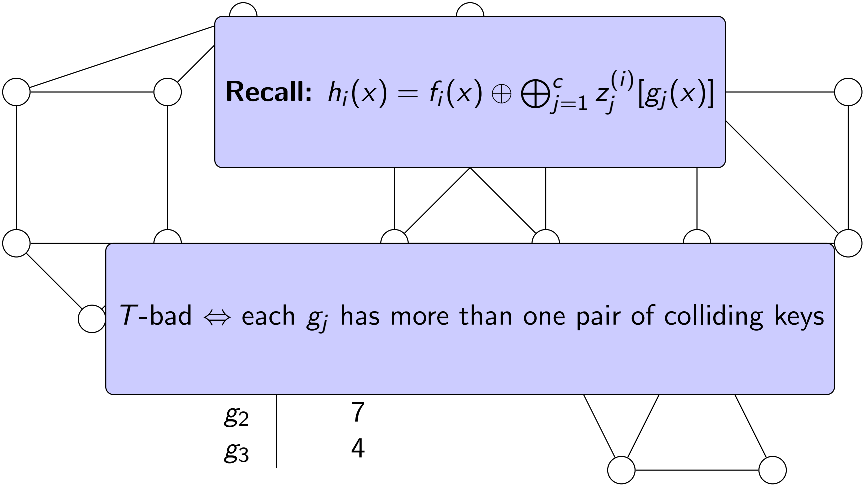




	#collisions
g_1	5
g_2	7
g_3	4

Peeling of bad graphs (simplified)

Assume "$\exists T \subseteq S : G(T, h_1, h_2)$ forms a $\mathrm{MOS}_s \cap (h_1, h_2)$ are T-bad ".



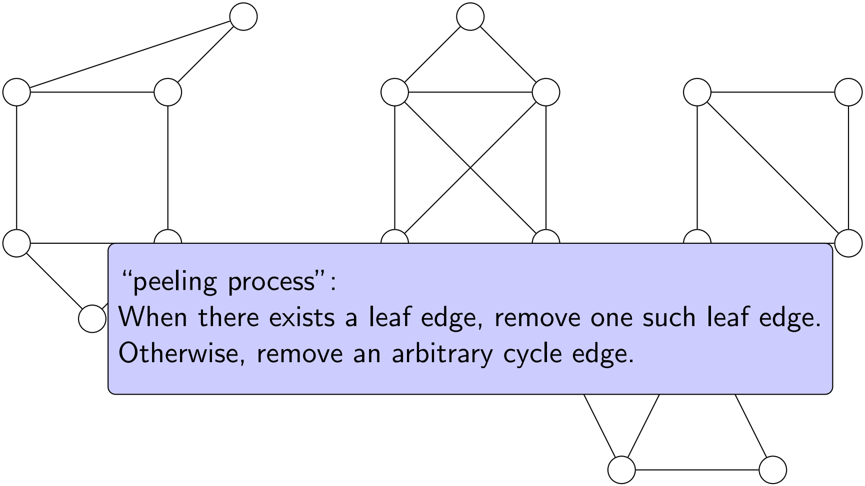
Recall: $h_i(x) = f_i(x) \oplus \bigoplus_{j=1}^{c} z_j^{(i)}[g_j(x)]$

T-bad $\Leftrightarrow$ each g_j has more than one pair of colliding keys

g_2	7
g_3	4

Peeling of bad graphs (simplified)

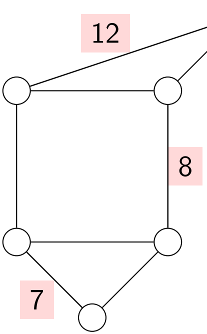
Assume "$\exists T \subseteq S : G(T, h_1, h_2)$ forms a $\text{MOS}_s \cap (h_1, h_2)$ are T-bad".

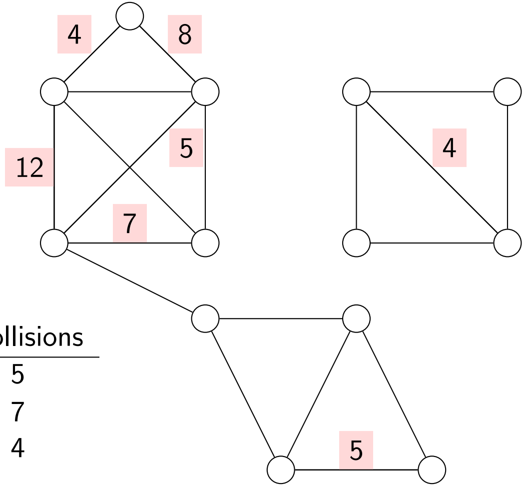


"peeling process":
When there exists a leaf edge, remove one such leaf edge.
Otherwise, remove an arbitrary cycle edge.

Peeling of bad graphs (simplified)

Assume "$\exists T \subseteq S : G(T, h_1, h_2)$ forms a MOS_s $\cap\, (h_1, h_2)$ are T-bad ".

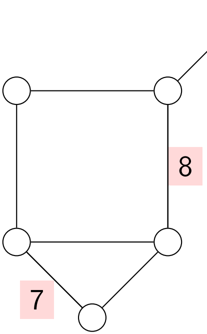


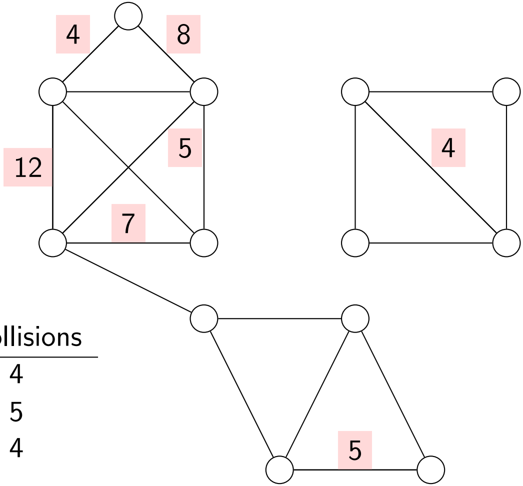


	#collisions
g_1	5
g_2	7
g_3	4

Peeling of bad graphs (simplified)

Assume "$\exists T \subseteq S : G(T, h_1, h_2)$ forms a MOS_s $\cap\, (h_1, h_2)$ are T-bad ".

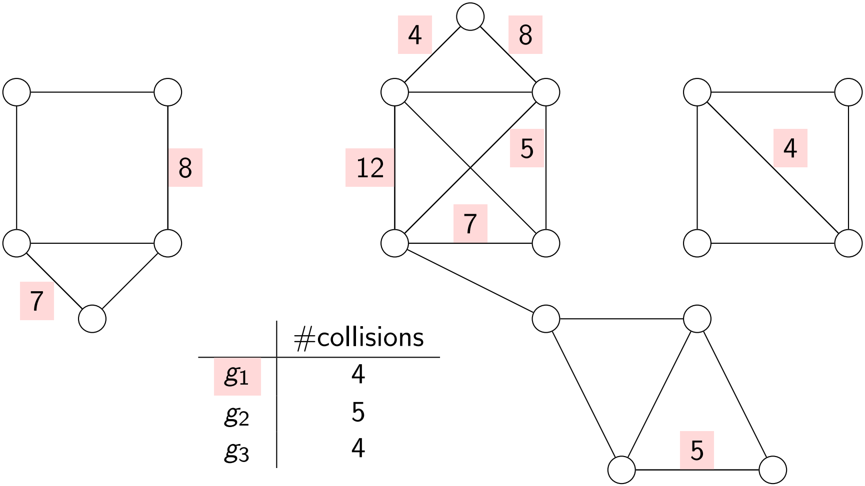




	#collisions
g_1	4
g_2	5
g_3	4

Peeling of bad graphs (simplified)

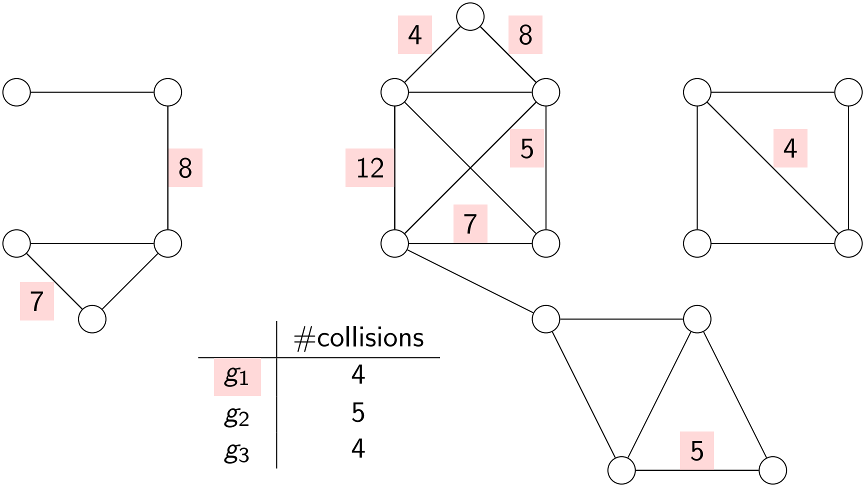
Assume "$\exists T \subseteq S : G(T, h_1, h_2)$ forms a MOS_s $\cap$ (h_1, h_2) are T-bad ".



	#collisions
g_1	4
g_2	5
g_3	4

Peeling of bad graphs (simplified)

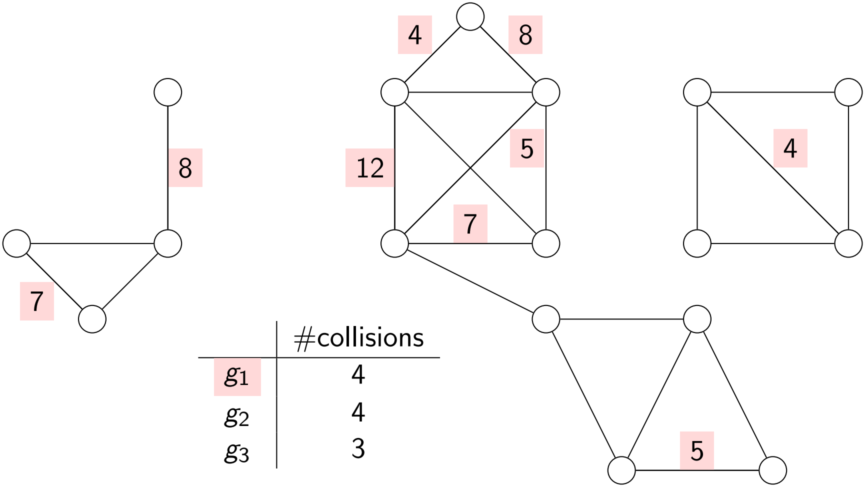
Assume "$\exists T \subseteq S : G(T, h_1, h_2)$ forms a MOS_s $\cap$ (h_1, h_2) are T-bad ".



	#collisions
g_1	4
g_2	5
g_3	4

Peeling of bad graphs (simplified)

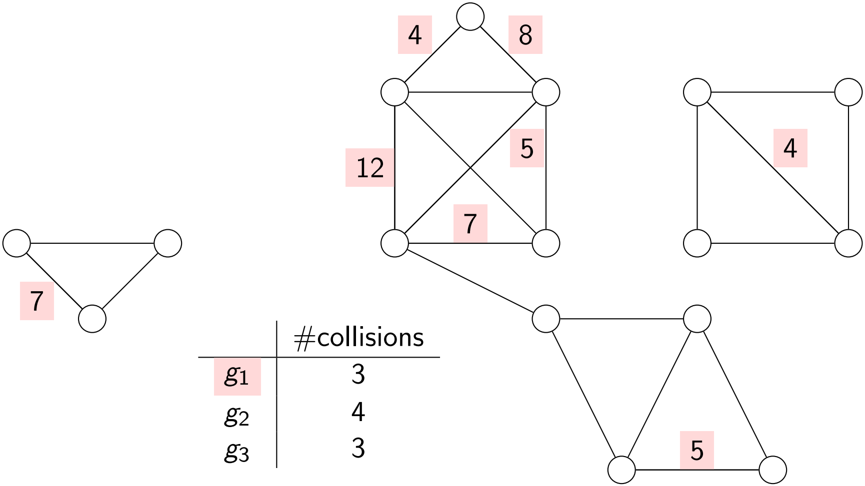
Assume "$\exists T \subseteq S : G(T, h_1, h_2)$ forms a $\mathrm{MOS}_s \cap (h_1, h_2)$ are T-bad ".



	#collisions
g_1	4
g_2	4
g_3	3

Peeling of bad graphs (simplified)

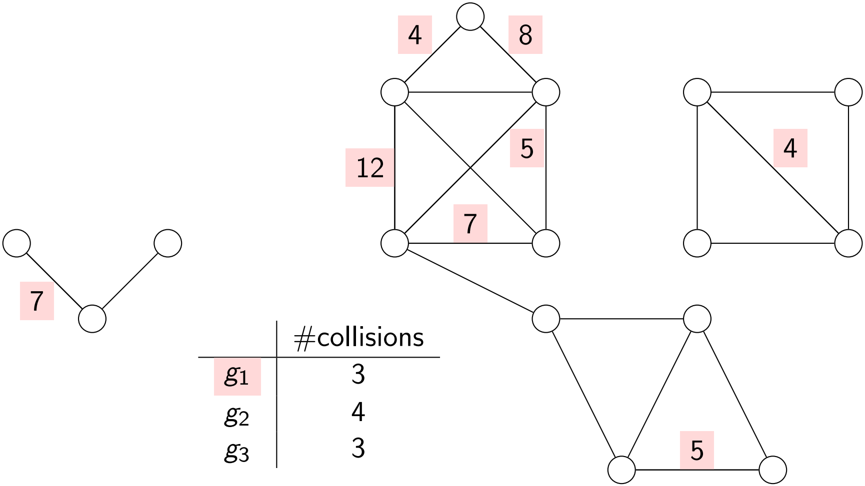
Assume "$\exists T \subseteq S : G(T, h_1, h_2)$ forms a MOS_s $\cap\, (h_1, h_2)$ are T-bad ".



	#collisions
g_1	3
g_2	4
g_3	3

Peeling of bad graphs (simplified)

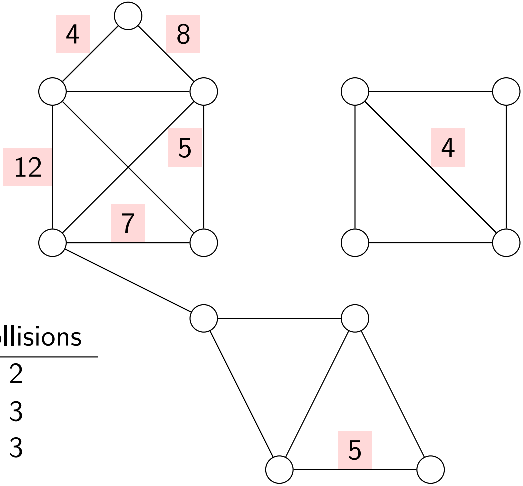
Assume “$\exists T \subseteq S : G(T, h_1, h_2)$ forms a MOS_s $\cap\, (h_1, h_2)$ are T-bad ”.

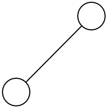


	#collisions
g_1	3
g_2	4
g_3	3

Peeling of bad graphs (simplified)

Assume "$\exists T \subseteq S : G(T, h_1, h_2)$ forms a $\text{MOS}_s \cap (h_1, h_2)$ are T-bad ".

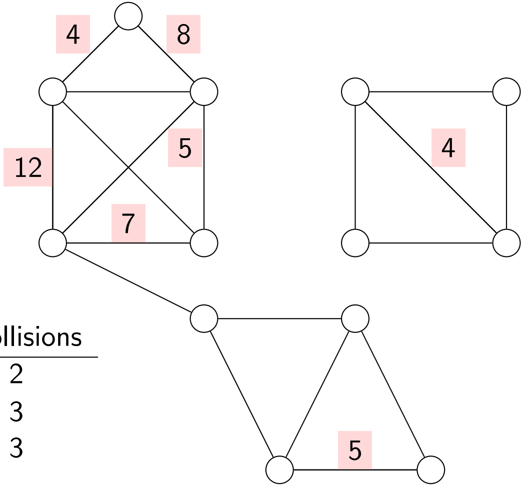




	#collisions
g_1	2
g_2	3
g_3	3

Peeling of bad graphs (simplified)

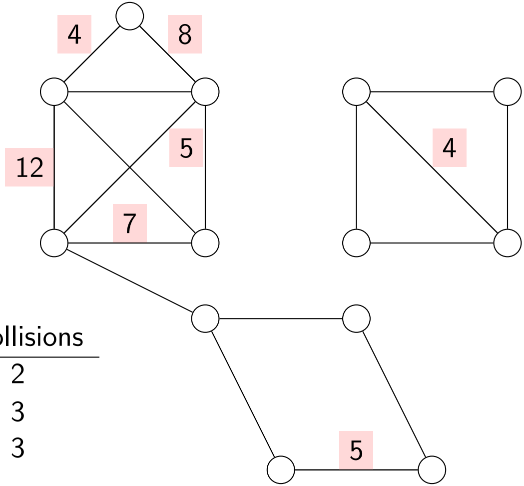
Assume "$\exists T \subseteq S : G(T, h_1, h_2)$ forms a $\text{MOS}_s \cap (h_1, h_2)$ are T-bad ".



	#collisions
g_1	2
g_2	3
g_3	3

Peeling of bad graphs (simplified)

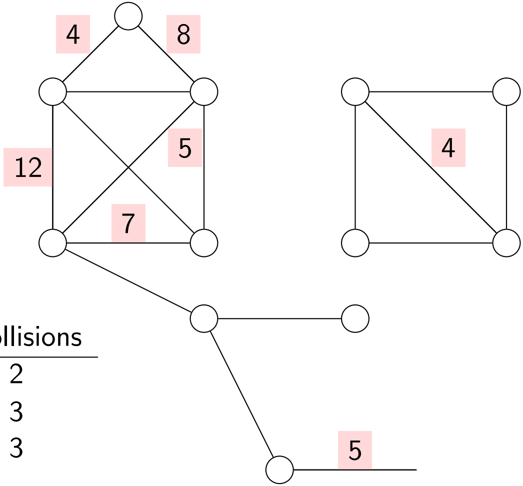
Assume "$\exists T \subseteq S : G(T, h_1, h_2)$ forms a $\mathrm{MOS}_s \cap (h_1, h_2)$ are T-bad"



	#collisions
g_1	2
g_2	3
g_3	3

Peeling of bad graphs (simplified)

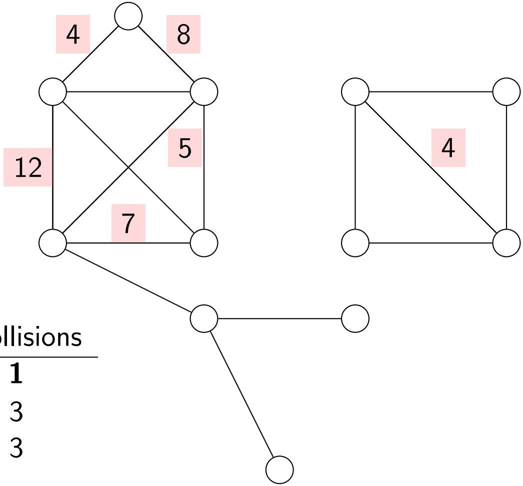
Assume "$\exists T \subseteq S : G(T, h_1, h_2)$ forms a $\text{MOS}_s \cap (h_1, h_2)$ are T-bad "



	#collisions
g_1	2
g_2	3
g_3	3

Peeling of bad graphs (simplified)

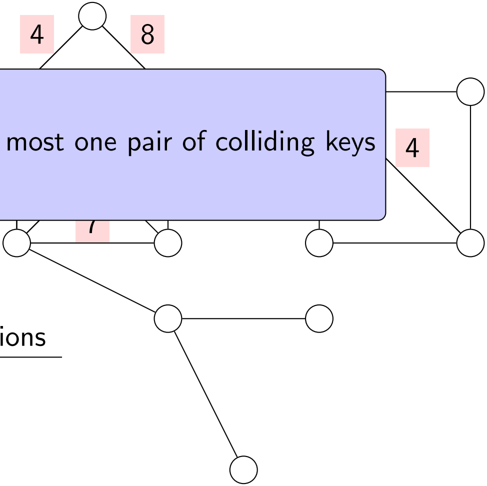
Assume "$\exists T \subseteq S : G(T, h_1, h_2)$ forms a $\text{MOS}_s \cap (h_1, h_2)$ are T-bad "



	#collisions
g_1	**1**
g_2	3
g_3	3

Peeling of bad graphs (simplified)

If "$\exists T \subseteq S : G(T, h_1, h_2)$ forms a MOS_s $\cap\,(h_1, h_2)$ are T-bad"



T-good $\Leftrightarrow$ exists g_j with at most one pair of colliding keys

	#collisions
g_1	**1**
g_2	3
g_3	3

then "$\exists T' \subseteq S : G(T', h_1, h_2)$ forms "peeled graph" $\cap\,(h_1, h_2)$ are T'-good"

Result of Peeling

$$\Pr(\exists T \subseteq S : G(T, h_1, h_2) \text{ forms a } \mathrm{MOS}_s \ \cap (h_1, h_2)\ T\text{-bad })$$
$$\leq \Pr(\exists T' \subseteq S\colon G(T', h_1, h_2) \text{ is peeling result} \cap (h_1, h_2)\ T'\text{-good})$$

- Can again use first moment method

Result of Peeling

$$\Pr(\exists T \subseteq S : G(T, h_1, h_2) \text{ forms a } \mathrm{MOS}_s \ \cap (h_1, h_2)\ T\text{-bad })$$
$$\leq \Pr(\exists T' \subseteq S\colon G(T', h_1, h_2) \text{ is peeling result} \cap (h_1, h_2)\ T'\text{-good})$$

- Can again use first moment method
- counting argument: at most

$$t^{O(\ell + c + \gamma)}$$

(unlabeled) graphs with t edges, ℓ leaf edges, c connected components, cyclomatic number γ.

Result of Peeling

$$\Pr(\exists T \subseteq S : G(T, h_1, h_2) \text{ forms a MOS}_s \ \cap (h_1, h_2) \ T\text{-bad })$$
$$\leq \Pr(\exists T' \subseteq S\colon G(T', h_1, h_2) \text{ is peeling result} \cap (h_1, h_2) \ T'\text{-good})$$

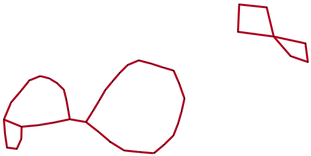
- Can again use first moment method
- counting argument: at most

$$t^{O(\ell+c+\gamma)}$$

  (unlabeled) graphs with t edges, ℓ leaf edges, c connected components, cyclomatic number γ.

Before/**After** peeling:

- $\ell = 0$
- $c \leq s + 1$
- $\gamma \leq 2(s + 1)$
- $|E| = |V| + s + 1$



Result of Peeling

$$\Pr(\exists T \subseteq S : G(T, h_1, h_2) \text{ forms a } \mathrm{MOS}_s \cap (h_1, h_2)\ T\text{-bad})$$
$$\leq \Pr(\exists T' \subseteq S \colon G(T', h_1, h_2) \text{ is peeling result} \cap (h_1, h_2)\ T'\text{-good})$$

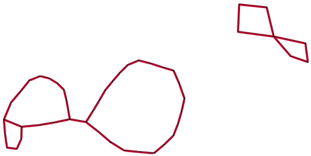
- Can again use first moment method
- counting argument: at most

$$t^{O(\ell + c + \gamma)}$$

  (unlabeled) graphs with t edges, ℓ leaf edges, c connected components, cyclomatic number γ.

Before/**After** peeling:

- $\ell = 0$, $\ell \leq 2$
- $c \leq s + 1$, $c \leq s + 1$
- $\gamma \leq 2(s + 1)$, $\gamma \leq 2s + 1$
- $|E| = |V| + s + 1$, $|E| \geq |V| - 1$



Result of Peeling /2

$$\Pr(\exists T \subseteq S : G(T, h_1, h_2) \text{ forms a } \mathrm{MOS}_s \ \cap (h_1, h_2)\ T\text{-bad}\ )$$
$$\leq \Pr(\exists T' \subseteq S\colon\ G(T', h_1, h_2) \text{ is peeling result} \cap (h_1, h_2)\ T'\text{-good})$$

- resulting graphs are sparser $\rightarrow$ they are more likely to occur
- use: when process stops each $g_j, 1 \leq j \leq c$, has a colliding pair of keys
- probability boost of $\approx (1/\sqrt{n})^c$
- probability of B^{MOS_s} is $O(n/\sqrt{n}^c)$, which is $1/n^{s+1}$ for $c = \Theta(s)$

Some applications need an additional "reduction step".
(Preserve collisions, make graphs smaller.)

Results

Simple graphs:

Results

Simple graphs:

- Cuckoo hashing (with a stash) $\rightarrow$ match fully random case.

Results

Simple graphs:

- Cuckoo hashing (with a stash) $\rightarrow$ match fully random case.
- Applications which need that largest component is $O(\log n)$ w.h.p. $\rightarrow m \geq 2.7n$. (Fully random case: $m \geq (1+\varepsilon)n$.)

Results

Simple graphs:

- Cuckoo hashing (with a stash) $\rightarrow$ match fully random case.
- Applications which need that largest component is $O(\log n)$ w.h.p. $\rightarrow m \geq 2.7n$. (Fully random case: $m \geq (1+\varepsilon)n$.)
- Simulation of a uniform hash function (Pagh/Pagh): simple construction.

Results

Simple graphs:

- Cuckoo hashing (with a stash) $\rightarrow$ match fully random case.
- Applications which need that largest component is $O(\log n)$ w.h.p. $\rightarrow m \geq 2.7n$. (Fully random case: $m \geq (1+\varepsilon)n$.)
- Simulation of a uniform hash function (Pagh/Pagh): simple construction.
- Constructing a perfect hash function (Bothelo/Pagh/Ziviani, 2013): constant number of constructions in expectation

Results

Simple graphs:

- Cuckoo hashing (with a stash) $\rightarrow$ match fully random case.
- Applications which need that largest component is $O(\log n)$ w.h.p. $\rightarrow m \geq 2.7n$. (Fully random case: $m \geq (1+\varepsilon)n$.)
- Simulation of a uniform hash function (Pagh/Pagh): simple construction.
- Constructing a perfect hash function (Bothelo/Pagh/Ziviani, 2013): constant number of constructions in expectation

Hypergraphs:

Results

Simple graphs:

- Cuckoo hashing (with a stash) $\rightarrow$ match fully random case.
- Applications which need that largest component is $O(\log n)$ w.h.p. $\rightarrow m \geq 2.7n$. (Fully random case: $m \geq (1+\varepsilon)n$.)
- Simulation of a uniform hash function (Pagh/Pagh): simple construction.
- Constructing a perfect hash function (Bothelo/Pagh/Ziviani, 2013): constant number of constructions in expectation

Hypergraphs:

- Parallel/Sequential Load Balancing: basically match bounds from fully random case (Schickinger/Steger, 2000).

Results

Simple graphs:

- Cuckoo hashing (with a stash) $\rightarrow$ match fully random case.
- Applications which need that largest component is $O(\log n)$ w.h.p. $\rightarrow m \geq 2.7n$. (Fully random case: $m \geq (1+\varepsilon)n$.)
- Simulation of a uniform hash function (Pagh/Pagh): simple construction.
- Constructing a perfect hash function (Bothelo/Pagh/Ziviani, 2013): constant number of constructions in expectation

Hypergraphs:

- Parallel/Sequential Load Balancing: basically match bounds from fully random case (Schickinger/Steger, 2000).
- Generalized cuckoo hashing (≥ 3 hash functions, $\ell \geq 2$ keys per cell): Some bounds by parallel load balancing: rather weak ($\approx 30\%$ space utilization), starting from $\ell \geq 8$.

Conclusion

We have seen:

- a generic framework to study randomness properties of graphs built using hash functions
- a class of hash functions that behaves well (= like a fully random hash function) on many interesting graph properties
- some applications of this hash class

Open:

- better bounds for some applications?
- bounds beyond first moment method?