

Cuckoo Filter: Simplification and Analysis

David Eppstein

15th Scandinavian Symposium and Workshops
on Algorithm Theory (SWAT 2016)

Reykjavik, Iceland, June 2016

Context

Goal: Data structure for a set of n identifiers (keys)
drawn from a larger universe of U potential identifiers

Want fast membership queries, small memory footprint

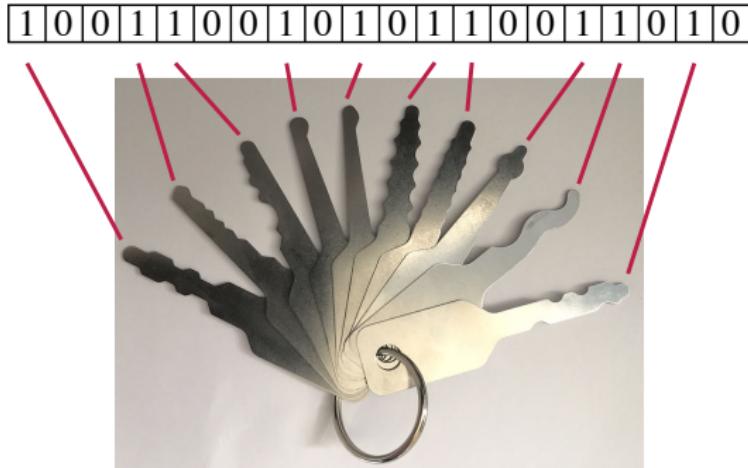
Other options (insert, delete, union, intersect) also useful



[File:Wafer Lock Try-Out Keys.jpg](#) by Willh26 on Wikimedia commons

Exact solutions: Bit vector

Store an array of bits, one per possible key
1 for set members, 0 for nonmembers



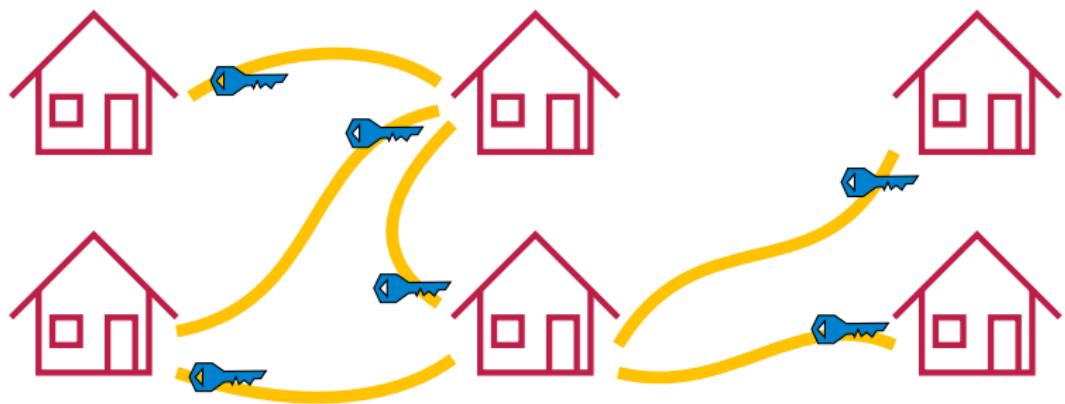
Fast queries, and vectorized union and intersection operations
But memory requirement $\Theta(U)$ is too large

Exact solutions: Cuckoo hashing (I)

[Pagh and Rodler 2004]

Each key is hashed to two home locations

Assign keys to homes and store one key per home



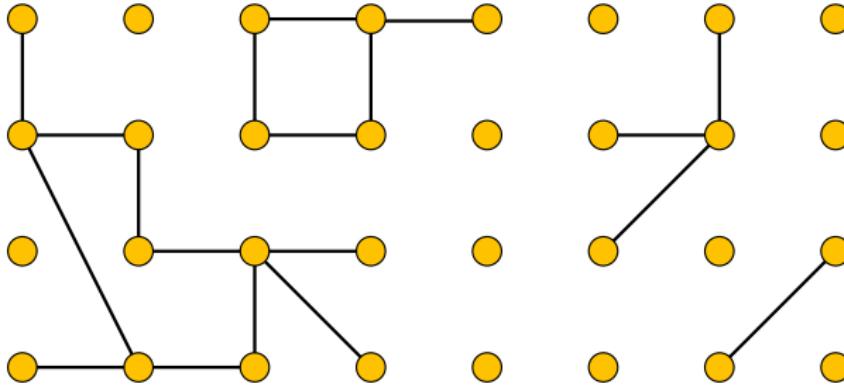
Constant worst-case query time (check both locations)

Constant average-case updates

Failure (unable to match keys to homes) has probability $O(1/n)$

Exact solutions: Cuckoo hashing (II)

Succeeds in matching keys to homes \iff the graph (homes, pairs selected by keys) is a *pseudoforest* (each component has ≤ 1 cycle)



Two weaknesses:

Failure probability of $O(1/n)$ may be too high

To achieve this, must leave $> 1/2$ of the homes empty
(too much wasted memory)

Exact solutions: Blocked cuckoo hashing

Store multiple keys/location [Dietzfelbinger and Weidling 2007]



Succeeds when no subset of location has too many keys

Allows near-optimal space $(1 + \epsilon)n \log_2 U$

Improves failure probability to $1/\text{polynomial}$ [Kirsch et al. 2010]

When even optimal space is too much

Reasons to use very little memory:

- ▶ Huge data sets,
too large to fit into main memory
- ▶ Small embedded devices
with little available memory
- ▶ Performance from fitting in cache

Solution: Approximate data structures!

Less memory but imprecise answers



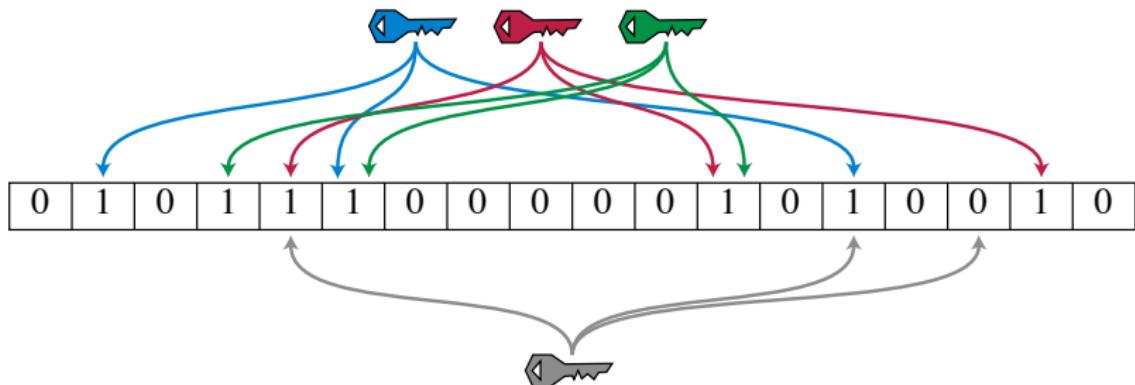
File:4856 - VIC-1211A Super Expander w 3k RAM open.JPG by Sven.petersen on Wikimedia commons

Approximate solutions: Bloom filter

[Bloom 1970]

Uses bitvector idea, but hashes each key to $O(1)$ bitvector cells

Query answer true \iff all hashed cells nonzero



A small number of keys that are not in the set will also have all cells nonzero – **false positives**

Uses $O(n \log 1/\rho)$ bits for false positive rate ρ

Bloom filters: enormously popular in practice

Google "bloom filter" 

Scholar About 14,700 results (0.05 sec)

Articles	Fast hash table lookup using extended bloom filter: an aid to network processing H Song, S Dhamapurikar, J Turner... - ACM SIGCOMM ..., 2005 - dl.acm.org Abstract Hash tables are fundamental components of several network processing algorithms and applications, including route lookup, packet classification, per-flow state management and network monitoring. These applications, which typically occur in the data-path of high- ... Cited by 302 Related articles All 17 versions Web of Science: 28 Import into BibTeX Save More	[PDF] from ut.ee UC-eLinks
Case law		
My library		

Any time	Space-code bloom filter for efficient per-flow traffic measurement A Kumar, J Xu, J Wang - Selected Areas in Communications, ..., 2006 - ieeeexplore.ieee.org Abstract—Per-flow traffic measurement is critical for usage accounting, traffic engineering, and anomaly detection. Previous methodologies are either based on random sampling (eg, Cisco's NetFlow), which is inaccurate, or only account for the “elephants.” We introduce a ... Cited by 247 Related articles All 18 versions Web of Science: 18 Import into BibTeX Save More	[PDF] from columbia.edu UC-eLinks
Since 2016		
Since 2015		
Since 2012		
Custom range...		

Sort by relevance	Less hashing, same performance: Building a better Bloom filter A Kirsch, M Mitzenmacher - Algorithms-ESA 2006, 2006 - Springer Abstract A standard technique from the hashing literature is to use two hash functions $h_1(x)$ and $h_2(x)$ to simulate additional hash functions of the form $g_i(x) = h_1(x) + i h_2(x)$. We demonstrate that this technique can be usefully applied to Bloom filters and related data ... Cited by 129 Related articles All 20 versions Web of Science: 25 Import into BibTeX Save More	[PDF] from astrometry.net UC-eLinks
Sort by date		

<input checked="" type="checkbox"/> include patents <input checked="" type="checkbox"/> include citations	Designing a Bloom filter for differential file access LL Gremillion - Communications of the ACM, 1982 - dl.acm.org Abstract The use of a differential file for a database update can yield integrity and performance benefits, but it can also present problems in providing current data to subsequent accessing transactions. A mechanism known as a Bloom filter can solve these ... Cited by 101 Related articles Web of Science: 17 Import into BibTeX Save More	UC-eLinks
<input type="checkbox"/> Create alert		

<input checked="" type="checkbox"/> include patents <input checked="" type="checkbox"/> include citations	An optimal Bloom filter replacement A Pagh, R Pagh, SS Rao - Proceedings of the sixteenth annual ACM- ..., 2005 - dl.acm.org Abstract This paper considers space-efficient data structures for storing an approximation S' to a set S such that $S \subseteq S'$ and any element not in S belongs to S' with probability at most ϵ . The Bloom filter data structure, solving this problem, has found widespread use. Our main ... Cited by 123 Related articles All 10 versions Import into BibTeX Save More	[PDF] from it-c.dk
<input type="checkbox"/> Create alert		

[HTML1 Space-efficient and exact de Bruijn graph representation based on a Bloom filter](#) [HTML1 from biomedcentral](#)

Drawbacks of Bloom filters

- ▶ Suboptimal memory
44% worse than lower bound
- ▶ Unable to delete items
(counting Bloom filter can but
uses $\omega(1)$ more memory)
- ▶ Poor memory access pattern
More accurate \Rightarrow more hits/query



File:2008 08 19 Einbreid Bru
Iceland.JPG by Crux on Wikimedia
commons

Better than Bloom filters

“An optimal Bloom filter replacement” [Pagh et al. 2005]

“Cuckoo filter: Practically better than Bloom” [Fan et al. 2014]



Both have optimal space, locality of reference, allow deletions

Pagh et al.: proven, but no practical implementation

Fan et al.: practical implementation but no proofs
... until now

Cuckoo filter main idea

Cuckoo hash, but save space by storing fingerprints instead of keys



Based on [File:Ninhydrin staining thumbprint.png](#) by Horoporo on Wikimedia commons

Answer query by checking whether the query key's fingerprint is at one of its homes

Complication: How to reshuffle keys after an insert?

In cuckoo hashing, homes are *independent* functions of key



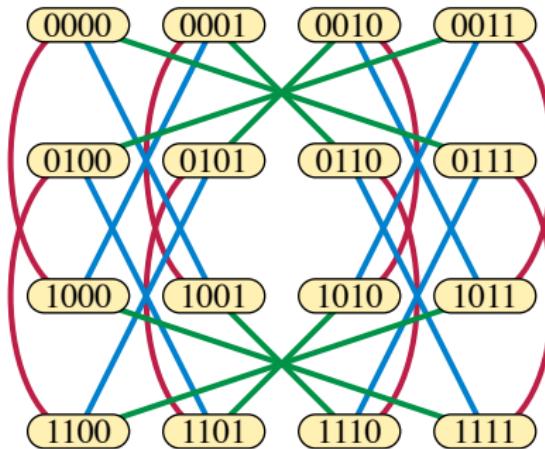
But cuckoo filter reshuffle only knows fingerprint+location, not key

Not enough information for second home to be independent

Solution: use $\text{hash}(\text{key})$ and $\text{hash}(\text{key}) \text{ xor } \text{hash}(\text{fingerprint})$

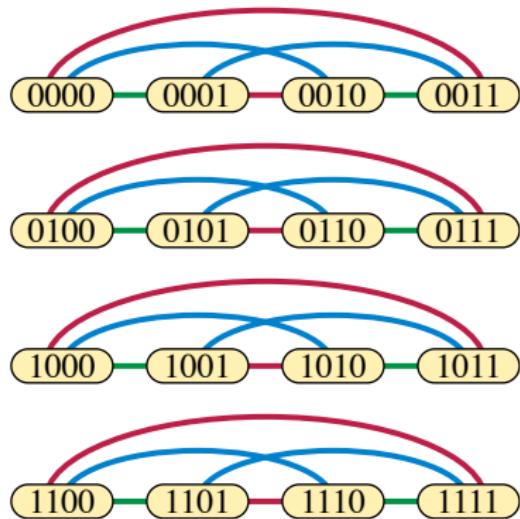
Simplification: $\text{hash}(\text{key})$ and $\text{hash}(\text{key}) \text{ xor } \text{fingerprint}$

Graph of pairs of homes for all fingerprints



Second home = first home
xor hash(fingerprint)

Colors show different
hash values

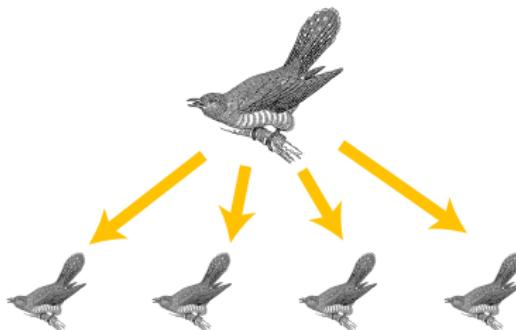


Second home = first home
xor fingerprint

Colors show different
(2-bit) fingerprints

Main ideas of analysis

When we use simplified home placement,
we are effectively partitioning the cuckoo filter
into many smaller cuckoo filters



The partition is highly likely to be well balanced
(standard argument using Chernoff bounds)

Within each of the smaller cuckoo filters,
pairs of homes are independent of each other
so we can use existing cuckoo hash analysis

Conclusions

The simplified cuckoo filter with sufficiently large constant b fingerprints/home and fingerprint size $f = \Omega((\log n)/b)$ can place all fingerprints with high probability

When it succeeds, it achieves false positive rate $\rho = O(b/2^f)$ using memory arbitrarily close to optimal, $(1 + \epsilon)n \log_2 1/\rho$ bits



File:Success sign.jpg by rmgimages from Wikimedia commons

Still open: Analyze cuckoo filtering without the simplification

References I

- Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970. doi: 10.1145/362686.362692.
- Martin Dietzfelbinger and Christoph Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. *Theoret. Comput. Sci.*, 380(1-2):47–68, 2007. doi: 10.1016/j.tcs.2007.02.054.
- Bin Fan, Dave G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. Cuckoo filter: Practically better than Bloom. In *Proc. 10th ACM Int. Conf. Emerging Networking Experiments and Technologies (CoNEXT '14)*, pages 75–88, 2014. doi: 10.1145/2674005.2674994.
- Adam Kirsch, Michael D. Mitzenmacher, and Udi Wieder. More robust hashing: cuckoo hashing with a stash. *SIAM J. Comput.*, 39(4):1543–1561, 2010. doi: 10.1137/080728743.

References II

- Anna Pagh, Rasmus Pagh, and S. Srinivasa Rao. An optimal Bloom filter replacement. In *Proc. 16th ACM-SIAM Symposium on Discrete Algorithms (SODA '05)*, pages 823–829. ACM, New York, 2005.
- Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004. doi: [10.1016/j.jalgor.2003.12.002](https://doi.org/10.1016/j.jalgor.2003.12.002).