Another Proof of Cuckoo Hashing with New Variants

Udi Wieder
VMware Research

June 5, 2018

Abstract

We show a new proof for the load of obtained by a Cuckoo Hashing data structure. Our proof is arguably simpler than previous proofs and allows for new generalizations. The proof first appeared in Pinkas *et al.* [PSWW18] in the context of a protocol for private set intersection. We present it here separately to improve its readability.

1 The Problem

In the Cuckoo Hashing scheme we are presented with n items $x_1, \ldots, x_n$ drawn from a finite universe $\mathcal{U}$. The goal is to store them in a hash table. To that end there are two arrays A_0, A_1, each with m slots. A memory slot can contain one item. We also have two hash functions $h_\sigma : \mathcal{U} \to [m]$, $\sigma \in \{0, 1\}$, where each function takes an item x_i and returns an index in $[m]$. A *legal placement* of the items is a mapping of each x_i either to $A_0[h_0(x_i)]$ or to $A_1[h_1(x_i)]$, such that every slot is assigned at most one item. The problem we aim to solve is the following: assuming the hash functions are drawn uniformly at random from the set of functions $\mathcal{U} \to [m]$, how small could m be while still guaranteeing the existence of a legal placement with high probability. We will prove the following theorem:

Theorem 1.1. *If the hash functions are drawn uniformly from the set of all functions $\mathcal{U} \to [m]$ and $m \geq (1+\epsilon)n$ for $\epsilon > 0$, then the probability there is a legal placement is $1 - O(1/n)$ where the big O notation hides constants depending on ϵ.*

1.1 d-dimensional Cuckoo Hashing

The motivation for coming up with a new proof stemmed from the need to find bounds for a generalization we call d-dimensional cuckoo hashing. In this case each hash function maps an element from the universe to a d-dimensional vector in $[m]^d$. The goal is to place d copies of an item, either in the d locations in A_0 indexed by h_0, or in the d locations in A_1 indexed by h_1. The case $d = 1$ corresponds to standard Cuckoo Hashing. Our proof below generalizes to this case and shows that it suffices to have $m \geq (1+\epsilon)d^2n$. See [PSWW18] for the original motivation and applications of this result.

1.2 Previous Approaches and Related Work

Theorem 1.1 was first proved in [PR04], other proofs exist, see the survey [Wie17]. We note that here we are concerned merely with the *existence* of a placement while previous proofs bound the running time of an algorithm that *finds* a legal placement. It is not hard to see that our approach could be used to that end as

well, however to the best of our knowledge it does not offer new insights or simplifications. We discuss that further in Section 3.2.

Previous proofs use the notion of a *Cuckoo Graph*. A cuckoo graph is a bipartite graph with m vertices on each side. Each vertex represents a memory slot and edges represent items. One can show that a legal placement exists iff every connected component in the cuckoo graph has at most one cycle. There are several approaches for showing that. The proof in [PR04] finds a small set of 'forbidden graphs' and shows by enumeration that with high probability none of them appear. Alternatively, one can use techniques from random graph theory [KMW09]. See survey [Wie17].

We differ by looking at a different graph which we call the *inference graph*. The inference graph tracks the logical constraints behind placement decisions, similar to inference graphs for 2-SAT. We show that a legal placement exists if and only if the inference graph does not contain a cycle. We then proceed to show via enumeration that a cycle is not likely to occur.

2 The Inference Graph

Given a pair of functions $h_\sigma : \mathcal{U} \to [m]$, $\sigma \in \{0,1\}$ and a set of items S, the *Inference Graph* $G(S, h_0, h_1)$ is composed of the following:

nodes: The set of nodes is comprised of two sets $a_1^0, \ldots, a_n^0$ and $a_1^1, \ldots, a_n^1$. Semantically we think of node a_i^σ as representing the event that x_i was placed in A_σ.

edges: The edges of the graph are directed and represent inferences between the events: If $h_\sigma(x_i) = h_\sigma(x_j)$ then G has the directed edges $(a_i^\sigma, a_j^{1-\sigma})$ and $(a_j^\sigma, a_i^{1-\sigma})$. In words, the edges (a_i^0, a_j^1), (a_j^0, a_1^i) mean that we cannot place both x_i, x_j in A_σ: if x_i is placed in A_σ then x_j must be placed in $A_{1-\sigma}$ and vice versa.

Note that this graph is somewhat similar to the structure constructed for resolution proofs of $2-$SAT formulas.

2.1 Placement

The goal of the following definitions is to form conditions under which an item x_i could be placed. We then will show that these conditions hold w.h.p for all items. Denote by $G(a_i^\sigma)$ the set of vertices reachable from a_i^σ (including a_i^σ) in the inference graph. We may drop the σ in our notation as our claims hold for both $\sigma = 0$ and $\sigma = 1$. When we refer to the items of $G(a_i)$, we mean all items associated with nodes of $G(a_i)$, that is: $\{x_j : a_j \in G(a_i)\}$.

The first observation to make is that if a node $a_j^\gamma \in G(a_i^\sigma)$ then any legal placement in which x_i is placed in table A_σ must have x_j placed in A_γ.

Definition 1. *A node a_i^σ in the Inference Graph is called* bad *if there is a j such that both $a_j^0, a_j^1 \in G(a_i^\sigma)$. An item x_i is* bad *if both a_i^0 and a_i^1 are bad.*

Namely, a node a_i^σ is bad if there is an item x_j such that placing x_i in table A_σ prevents placing x_j in either table A_0 or table A_1. So there is no legal placement in which x_i is placed in A_σ. An item x_i is bad if placing it in either table A_0, A_1 prevents finding a placement for other items.

Clearly, a bad item implies that not all items could be placed. The following lemma states the converse is also true.

Lemma 2.1. *If node a_i^σ is not bad then all items of $G(a_i^\sigma)$ could be placed.*

Proof. We first place x_i in A_σ. Then place all its neighbors in $A_{1-\sigma}$ and continue iteratively. Note that a node associated with an occupied slot is part of $G(a_i)$. Now, if item x_j cannot be placed then it must intersect items both on A_0 and on A_1 which means both a_j^0 and a_j^1 are in $G(a_i)$ which is a contradiction. □

Lemma 2.2. *If none of the items are bad then all items could be placed in the tables.*

Proof. The algorithm that places all the items is now straightforward: Let S be the set of currently unplaced items. Pick an item $x_i \in S$ and since it is not bad, then a_i^σ is not bad for some $\sigma \in \{0,1\}$. Now by Lemma 2.1 all items of $G(a_i)$ could be placed successfully. Let S' be the remaining items, i.e., $S' = S \setminus G(x_i)$. Given S' and all the free locations in A_0, A_1 we compute the new inference graph $G' = G(h_0, h_1, S')$ and continue inductively. The only thing remaining to observe is that if there were no bad items in G then there are no bad items in G'. To see this observe that G' is a subgraph of G. Indeed let x_j be an item in G'. Note that both slots $h_\sigma(x_j)$ must be free, otherwise $a^\sigma \in G(a_i)$, so every inference made in G' is true also for G. □

2.2 Main Result

The goal now is to calculate the probability an item is bad.

Theorem 2.3. *If the size of each table is greater than $m = (1+\epsilon)n$ then for every item i, the probability x_i is bad is at most $\left(\frac{1+\epsilon}{\epsilon}\right)^3 \cdot \frac{2}{m^2}$, where the probability is taken over the choice of the hash functions.*

Taking a union bound over all x_i proves Theorem 1.1. The remainder of the section is dedicated to the proof Theorem 2.3.

Our approach is to show that it is unlikely that an item is bad. For that to happen both its nodes need to be bad, and we would like to count how many bad graphs are there and show that they are unlikely to appear. As is often the case in proofs based on counting argument, the trick is to carefully define the objects which we count. In order to facilitate this bound we need to constrain further the exact notion of a bad node, captured by the next definition:

Definition 2. *A* bad path *rooted at a_i^σ is a simple path from a_i^σ to $a_i^{1-\sigma}$. A bad path is called* basic *if it does not contain a bad path. In other words, for each $j \neq i$ at most one of $\{a_j^0, a_j^1\}$ can appear in the path.*

The next lemma shows that basic bad paths are the only type of subgraphs we need to care about.

Lemma 2.4. *If a node is bad then it is the root of a basic bad path.*

Proof. Assume a_i^σ is bad, there must be at least one j for which a_i^σ is connected to both a_j^0, a_j^1. Further, we can assume that there is no $k \neq j$ such that both a_k^0 and a_k^1 appear on the paths from a_i^σ to a_j^0, a_j^1. We can make this assumption because if there is, we may take the pair a_k^0, a_k^1 instead.

Now recall that by construction, if an edge $(a_k^\sigma, a_\ell^{1-\sigma})$ appears in the inference graph, then so does the edge $(a_\ell^\sigma, a_k^{1-\sigma})$. A simple induction shows that if there is a path $a_i^\sigma \rightsquigarrow a_j^{1-\sigma}$ then there is a path $a_j^\sigma \rightsquigarrow a_i^{1-\sigma}$. Thus, we can construct a path $a_i^\sigma \rightsquigarrow a_j^\sigma \rightsquigarrow a_i^{1-\sigma}$. Further, there is no item which is repeated in the path. □

We now need to show that the probability an item is bad is small. We start by bounding the probability a node is the root of a basic bad path. Recall that m denotes the number of slots in each array.

Lemma 2.5. *If $m \geq (1+\epsilon)n$ then for every item i, the probability a_i^0 is the root of a basic bad path is at most $\frac{1+\epsilon}{\epsilon m}$.*

Proof. We count the number of labeled basic bad paths rooted at a_i^0. Let k be the number of edges in the path, and $k+1$ the number of nodes. The head of the path is already set to be a_i^0, and the tail is a_i^1 so there are at most n^{k-1} ways of choosing the nodes in between. We conclude:

$$\#\text{possible labeled basic bad paths rooted at } a_i^0 \leq \sum_k n^{k-1} \tag{1}$$

Given a labeling of a bad path, we calculate the probability it actually appears in the graph. Consider an edge in the path $(a_k^\sigma, a_\ell^{1-\sigma})$. This edge belongs to the inference graph iff $h_\sigma(x_k) = h_\sigma(x_\ell)$ which happens[1] with probability $\leq 1/m$. Now, since no item in the path is repeated, the occurrences of the k edges are independent events. The probability a given path appears in the graph is therefore $\leq 1/m^k$.

Combined with (1) we have that the probability a_i^0 is the root of a bad path is at most

$$\frac{1}{m}\sum_{k\geq 1}\left(\frac{n}{m}\right)^{k-1} \leq \frac{1}{m}\sum_{k\geq 1}\left(\frac{1}{1+\epsilon}\right)^{k-1} \leq \frac{1+\epsilon}{\epsilon m} \tag{2}$$

□

Proof of Theorem 2.3. Note that we bounded the probability node a_i^0 is bad. For item x_i to be bad node a_i^1 has to be bad as well. Again, it is enough to bound the case a_i^1 is the root of a basic bad path. So we condition on a_i^0 having a basic bad path. When accounting for all possible basic bad paths rooted at a_i^1 we need to differentiate between the various ways in which they intersect the basic bad path rooted in a_i^0. We proceed via a small case analysis.

Case 1**:** Assume the items associated with the bad path do not intersect those of the path from a_i^0. In this case the conditioning on the path from a_i^0 has no affect; the same calculation holds as before and the probability item x_i is bad is at most $\left(\frac{1+\epsilon}{\epsilon m}\right)^2$.

Case 2**:** There is some item x_j which belongs to both bad paths rooted at a_i^0 and a_i^1. Let k_1 be the length of the bad path starting at a_i^0 and k_2 be the length of the prefix of the bad path starting at a_i^1 and ends at the intersection: a_j^σ. Note that there are at most k_1 possibilities for choosing x_j. The probability such a path exists is therefore at most

$$\begin{aligned}
&\frac{1}{m}\sum_{k_1\geq 1}\left(\frac{1}{1+\epsilon}\right)^{k_1-1}\frac{2k_1}{m}\sum_{k_2\geq 1}\left(\frac{1}{1+\epsilon}\right)^{k_2-1} \\
\leq &\frac{2(1+\epsilon)}{\epsilon m^2}\sum_{k_1\geq 1} k_1\left(\frac{1}{1+\epsilon}\right)^{k_1-1} \\
\leq &\left(\frac{1+\epsilon}{\epsilon}\right)^3\cdot\frac{2}{m^2}
\end{aligned}$$

□

Taking a union bound over all items we have:

Corollary 2.6. *If $m \geq (1+\epsilon)n$ then the probability there is a failure is at most $\frac{2(1+\epsilon)^2}{\epsilon^3}\cdot\frac{1}{n}$.*

[1]This is the only place we need to augment the proof for the d-dimensional case, where the probability is roughly d^2/m

3 Remarks

3.1 Generalization to multi-dimensional cuckoo hasing

What is there to gain from a new proof? it is our subjective view that this proof is simpler and easier to follow, at least as an offline result. However, the main use of a new proof to a known result is to increase the understanding of the result and hopefully generalize in a new direction. Indeed the motivation for this work stemmed from a new and different application of a Cuckoo Hashing variant used in protocols for private set intersection. In this variant which we call d-dimensional cuckoo hashing, each hash function maps an element from the universe to a d-dimensional vector in $[m]^d$. The goal is to place d copies of an item, either in the d locations in A_0 indexed by h_0, or in the d locations in A_1 indexed by h_1. The case $d = 1$ corresponds to standard Cuckoo Hashing. It is straightforward to see that the exact same proof holds, the only difference being in equation 2 where now the probability an edge appears in the path is d^2/m. See [PSWW18] for the original motivation and application of this result.

3.2 Online vs Offline and Running time

Note that in this manuscript the problem tackled is the existence if a legal placement, without an explicit argument to find it. Of course, the proof in effect describes an insertion algorithm but does not argue anything about its running time and further, assumes that the entire graph is given in advance. A major advantage of Cuckoo Hashing however is that the dynamic insertion algorithm, as described in [PR04] is efficient, namely takes $O(1)$ on expectation and $O(\log n)$ w.h.p.. In previous proofs the running time of the insertion algorithm is analyzed via a careful analysis of the structure of the cuckoo graph, in particular, the running time is bounded via a bound on the size of its connected component. A similar type of argument could be made here for the inference graph as well. It offers no new simplifications or generalization (to the best of our understanding) so we saw no point in spelling it out.

3.3 Stash

In [KMW09] it is shown that augmenting the scheme with s extra slots that can hold any item (a.k.a stash) reduces the probability of insertion failure to roughly $n^{-(s+1)}$. A more combinatorial proof was given in [ADW12]. See also Theorem 5.5 in [Wie17]. A similar argument could be made here, but we do not think it offers simplifications or new insights.

Acknowledgments

I'm indebted to my co-authors of [PSWW18]: Thomas Schneider, Christian Weinert, and especially Benny Pinkas.

References

[ADW12] Martin Aumüller, Martin Dietzfelbinger, and Philipp Woelfel. Explicit and efficient hash families suffice for cuckoo hashing with a stash. In *Proceedings of the 20th Annual European Conference on Algorithms*, ESA'12, pages 108–120, Berlin, Heidelberg, 2012. Springer-Verlag.

[KMW09] A. Kirsch, M. Mitzenmacher, and U. Wieder. More robust hashing: Cuckoo hashing with a stash. *SIAM J. Comput.*, 39(4):1543–1561, 2009.

[PR04] R. Pagh and F. F. Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004.

[PSWW18] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based psi via cuckoo hashing. In *Advances in CryptologyEUROCRYPT*, 2018.

[Wie17] Udi Wieder. Hashing, load balancing and multiple choice. *Foundations and Trends in Theoretical Computer Science*, 12(34):275–379, 2017.