Peeling Arguments and Double Hashing

Michael Mitzenmacher[1] and Justin Thaler[2]

Abstract— **The analysis of several algorithms and data structures can be reduced to the analysis of the following greedy "peeling" process: start with a random hypergraph; find a vertex of degree at most k, and remove it and all of its adjacent hyperedges from the graph; repeat until there is no suitable vertex. This specific process finds the k-core of a hypergraph, and variations on this theme have proven useful in analyzing for example decoding from low-density parity-check codes, several hash-based data structures such as cuckoo hashing, and algorithms for satisfiability of random formulae. This approach can be analyzed several ways, with two common approaches being via a corresponding branching process or a fluid limit family of differential equations.**

In this paper, we make note of an interesting aspect of these types of processes: the results are generally the same when the randomness is structured in the manner of double hashing. This phenomenon allows us to use less randomness and simplify the implementation for several hash-based data structures and algorithms. We explore this approach from both an empirical and theoretical perspective, examining theoretical justifications as well as simulation results for specific problems.

I. Introduction

The analysis of several algorithms and data structures can be reduced to the analysis of the following greedy "peeling" process: start with a random hypergraph; find a vertex of degree at most k, and remove it and all of its adjacent hyperedges from the graph; repeat until there is no suitable vertex. This specific process finds the k-core of a hypergraph. The k-core itself appears in the analysis of several algorithms, and variations on this underlying process have arisen in the analysis of many problems that on their surface appear quite different, such as low-density parity-check codes [16], cuckoo hashing [23], [5], [7], [9], and the satisfiability of random formulae [2], [18], [22].

An interesting question is whether the "full randomness" assumption for the underlying random graph is necessary for the analysis of the peeling process. Specifically, we consider here a hashing scheme that assigns d possible locations for a key x, such as for example cuckoo hashing, where the key will be placed in one of the d location choices. For any set of inputs S, the hashing scheme naturally defines a random hypergraph G, where nodes in G correspond to locations, and for each key $x \in S$ there is a hyperedge in G corresponding to the d location choices for x. Instead of computing values using d independent hash functions $h_1(x), h_2(x), \ldots, h_d(x)$, consider the following alternative: we compute two hash values $f_1(x)$ and $f_2(x)$, and then use as our d hash values $h_i(x) = f_1(x) + if_2(x) \bmod n$ for $i \in \{0, \ldots, d-1\}$, where n is the size of our hash table. (We generally assume that $f_1(x)$ is uniform over $[0, n-1]$, and $f_2(x)$ is uniform over all numbers in $[1, n-1]$ relatively prime to n to avoid duplicated hash values, and all hash values are independent.) This approach is commonly referred to as double hashing, and it has clear utility in the context of hash tables: it reduces the amount of randomness and computation required. Moreover, multiple-choice hashing is used in several hardware systems (such as routers), and double hashing is extremely conducive to implementation in hardware.

In the general setting of hypergraphs, we can think of double hashing restricting the hyperedges chosen so that the vertices in the hyperedge form an arithmetic sequence. We can view this informally as a particular type of "quasi-random" hypergraph, and the motivation from multiple-choice hashing schemes makes it a natural subclass of random hypergraphs to study. One might ask about the behavior of any number of graph properties for these double hashing hypergraphs. Here we suggest that it is particularly interesting to ask how the change from fully random hypergraphs to double hashing hypergraphs affects results like the computation of the k-core, given that the k-core plays a key role in the analysis several algorithms and data structures.

In this paper, we provide empirical evidence and theoretical justification that using double hashing does not affect the size of the k-core, except up to lower order terms. This suggests that double hashing may be suitable for use in a number of practical settings. We provide experimental results for the 2-core and 3-core, as well as for cuckoo hashing; we also consider how previous analyses, based on branching processes or differential equations, might extend to the double hashing setting.

Several interesting open questions remain, and we believe that these hypergraphs, which we have dubbed *double hashing hypergraphs*, are worthy of further study.

A. Previous Work

Recall that, for open address hash tables, double hashing works as follows. The jth ball obtains two hash values, $f(j)$ and $g(j)$. For a hash table of size n, $f(j) \in [0, n-1]$ and $g(j) \in [1, n-1]$. Successive locations $h(j,k) = f(j) + kg(j) \bmod n$, $k = 0, 1, 2, \ldots\ldots$, are tried until an empty slot is found. On an insertion, the ball is placed in that location; on a lookup, the locations are tried in that order under the ball is found, or if an empty slot is found the search in unsuccessful. Interestingly, the average length of an unsuccessful search sequence for standard double hashing

[1]School of Engineering and Applied Sciences, Harvard University. Supported in part by the NSF under grants CNS-1011840, IIS-0964473, and CCF-091592.

[2]School of Engineering and Applied Sciences, Harvard University. Supported by an NSF Graduate Research Fellowship.

when the table load is a constant α has been shown to be, up to lower order terms, $1/(1-\alpha)$, showing that double hashing has essentially the same performance as random probing (where each ball would have its own random permutation of the bins to examine, in order, until finding an empty bin) [1], [10], [17]. However, these results appear to have been derived using different techniques than we examine here.

The idea of using double hashing in place of random hashing for multiple choice schemes has appeared in prior work. A highly related setting where double hashing was shown to provide the same asymptotic performance as fully random hashing was in the context of Bloom filters, first in the empirical work of Dillinger and Manolios [6], and then later in a corresponding theoretical justification given by Kirsch and Mitzenmacher [12]. They showed in standard settings where the false positive probability is designed to be constant, the false positive rate achieved using double hashing equals the rate achieved using fully random hashing (up to lower order terms). More recently, Mitzenmacher has shown that in the setting of balanced allocation (hash tables with multiple choices), the fluid limit approach applies even when using only double hashing [20]. In hindsight, both of these works can be seen as exemplars of the more general framework we propose here.

In a more general sense, this work falls into the long history of work on reducing required randomness for hashing, perhaps the most notable of which is the seminal work of Carter and Wegman on universal hash functions [3]. More recently, Pătraşcu and Thorup study simple tabulation hashing in the context of several algorithms, including linear probing and cuckoo hashing [24]. The main goal of our work is to show that in many contexts the use of double hashing in lieu of random hashing leads to results that are essentially indistinguishable in practice, and to provide some partial theoretical justification for this phenomenon. Arguably, this is a stronger goal than most related work, which aims to show that results obtained using reduced randomness are good approximations to, or asymptotically the same as, results using full randomness.

B. Paper Outline

The remainder of the paper is structured as follows. In Section II, we adapt a branching argument for analyzing the k-core of a random hypergraph to the double hashing setting. This section contains the main technical content of the paper, including our primary conjecture (Conjecture 2) and our main theorem (Theorem 3, which proves Conjecture 2 for a limited range of parameters). In Section III, we describe a different analysis of the k-core of a random hypergraph based on differential equations; our intention here is to raise the question of whether these types of arguments can be extended to the case of double hashing. Section IV introduces cuckoo hashing and discusses its relationship to k-cores of random hypergraphs. Section V presents our simulation results, suggesting that in many settings double hashing hypergraphs yield results that are essentially identical to truly random hypergraphs.

II. The Branching Argument for k-Cores

There are various forms of the branching argument for analyzing the k-core of a random graph. We utilize a variation based primarily on [22], although similar arguments appear elsewhere (e.g., [5], [15]). Our main insight is that these branching arguments, for the most part, also apply when only double hashing is used to determine the random hyperedges.

Specifically, we review the analysis of the asymptotic size of the k-core of a random hypergraph with n hyperedges on m vertices where $c = n/m$ is held fixed as n and m grow, and then consider the related analysis for the double hashing setting. We can allow our random hypergraph to have hyperedges with varying numbers of vertices; let λ_j be the probability a hyperedge has degree j. We assume $\lambda_0 = \lambda_1 = 0$ and $\lambda_i = 0$ for all i greater than some constant value. We define $\lambda(x) = \sum_j \lambda_j x^j$. For convenience we assume all vertices in a hyperedge are distinct, although asymptotically the difference is negligible in our arguments. Also, we find it slightly easier to use the binomial random hypergraph formulation where each hyperedge of degree j is present with probability $\frac{n\lambda_j}{\binom{m}{j}}$; again, asymptotically, this is equivalent.

We can consider the peeling process as occurring in a sequence of rounds, where in each round all vertices of degree less than k are removed the graph, along with their incident hyperedges. We calculate the probability q_h that a vertex z is deleted after h rounds of peeling. We follow the description of [5]. Whether a node z remains after a constant number of iterations h of the peeling procedure depends only on the sub-hypergraph induced on the vertices at distance at most h from z. In the setting where n is linear in m, this subgraph is a hypertree with probability $1 - o(1)$, and we assume this henceforth. Consider the hypertree rooted at z; the children are vertices that share a hyperedge with z, and the children of one of these vertices correspond to other vertices that share an edge with that vertex, and so on. The vertices at distance $h-1$ from z are deleted if they have at most $k-2$ children; that is, they are adjacent to at most $k-1$ hyperedges. We then consider vertices at $h-2$ from z in the same manner, taking into account vertices that have already been deleted, and so on. The root z is deleted if at the end of this process it has degree at most $k-1$.

For $i < h$, let p_i be the probability that a vertex of distance $h-i$ from z is deleted within the first i rounds of the peeling process. Let $\text{Bin}(N,p)$ denote a random variable with a binomial distribution, and $\text{Po}(\beta)$ denote a Poisson distributed random variable with expectation β. We make use of the asymptotic Poisson approximation of the binomial distribution, which results in additive terms that tend to zero in the asymptotic setting; also, we use the fact that the sum of Poisson random variables is itself Poisson. We have that $p_0 = 0$ and that p_i follows the recursion:

$$\begin{aligned}p_{i+1} &= \Pr\left[\sum_j \mathrm{Bin}\left(\binom{m-1}{j-1}, \frac{n\lambda_j}{\binom{m}{j}}(1-p_i)^{j-1}\right) < k-1\right]\\ &= \Pr\left[\sum_j \mathrm{Po}\left(n\lambda_j(1-p_i)^{j-1}\frac{\binom{m-1}{j-1}}{\binom{m}{j}}\right) < k-1\right] + o(1)\\ &= \Pr\left[\sum_j \mathrm{Po}\left(cj\lambda_j(1-p_i)^{j-1}\right) < k-1\right] + o(1)\\ &= \Pr\left[\mathrm{Po}(c\lambda'(1-p_i)) < k-1\right] + o(1).\end{aligned}$$

Here the $(1-p_i)^{j-1}$ term in the first line represents that $j-1$ neighboring vertices of an edge must remain after i rounds for that edge to have survived for consideration in the next round of peeling. We have used the fact that the sum of Poisson random variables is itself Poisson, and the resulting expression nicely is in terms of the derivative $\lambda'(x)$.

We note q_h has a slightly different form, accounting for the node in question being the root:

$$q_h = \Pr\left[\mathrm{Po}(c\lambda'(1-p_{h-1})) < k\right] + o(1).$$

However, the probability that z is deleted approaches $p = \lim_{j\to\infty} p_j$, and p corresponds to the smallest non-negative solution of

$$p = \Pr\left[\mathrm{Po}(c\lambda'(1-p)) < k-1\right].$$

The above argument represents the core of the argument for finding the asymptotic size of the k-core in a random hypergraph. In particular, the above suggests that there are two key regimes to consider; when $c < c_k$, the k-core is empty with high probability (by which we shall mean $1-o(1)$ throughout, for convenience), and when $c > c_k$, the k-core is of size $pm + o(m)$ with high probability, where p is determined by the equation above. However, there are details that remain for both cases, as shown in [22]. First, when $c < c_k$, the above argument shows that for any constant ϵ, the peeling process leaves fewer than ϵm vertices after a constant number of rounds. One must then separately prove that it is not possible to have a core of size at most $\epsilon' m$ for some constant ϵ' with high probability, which then yields for this case that the k-core will be empty. Similarly, when $c > c_k$, the above argument shows that after any constant number of rounds, the peeling process leaves at least pm vertices; some additional work is needed to show that in fact the k-core does not fall below $pm + o(m)$ vertices even if one allowed more than a constant number of rounds.

We now consider what changes when we move to double hashing in place of random hashing. The key point is that the central argument, based on the branching process, continues to hold even when using double hashing. To see this, let γ_j be the total number of possible hyperedges of degree j that can be chosen using double-hashing. The set of possible degree j hyperedges is $E_j = \{(f_1(x), f_1(x)+f_2(x), \ldots, f_1(x) + (j-1)f_2(x) : f_1(x) \in \{0,\ldots,m-1\}, f_2(x) \in \{1,\ldots,m-1\}, \gcd(f_2(x), m) = 1\}$, and so $\gamma_j = |E_j| = \Omega(m^2)$.

Specifically, we note the following fact for when m is prime, which we draw on freely later.

Fact 1: *If m is prime and $m > j^2$, then the following properties hold.*

1) *There are exactly $\gamma_j = \binom{m}{2}$ length j arithmetic sequences (taken modulo m) that are subsets of $\{0,\ldots,m-1\}$.*
2) *Suppose we generate a random length j arithmetic sequence via double-hashing. Then all length-j arithmetic sequences are equally likely to be generated.*

Proof: Both statements follow by observing that there are exactly two ways to generate any arithmetic sequence of the form $e = \{a, a+b, \ldots a+(j-1)b\}$ by double hashing. Specifically, e will be generated if and only if $f_1(x) = a$ and $f_2(x) = b$, or $f_1(x) = a+(j-1)b$ and $f_2(x) = -b$. If m is prime, there are $m(m-1)$ ways to choose values $(f_1(x), f_2(x))$, and there are therefore $m(m-1)/2 = \binom{m}{2}$ possible length j arithmetic sequences. Moreover, each is equally likely to be generated. ■

We again use a binomial random hypergraph formulation, where each possible hyperedge available via double hashing of degree j is present with probability $\frac{n\lambda_j}{\gamma_j}$. Further, the subgraph of vertices within distance h from a specific vertex again forms a hypertree with probability $1-o(1)$, and we assume this henceforth. Now note that each vertex, by symmetry, appears in $\frac{\gamma_j j}{m}$ possible hyperedges. Hence, repeating the argument regarding the derivation of the equation for the p_i, we find in the double hashing setting:

$$\begin{aligned}p_{i+1} &= \Pr\left[\sum_j \mathrm{Bin}\left(\frac{\gamma_j j}{m}, \frac{n\lambda_j}{\gamma_j}(1-p_i)^{j-1}\right) < k-1\right]\\ &= \Pr\left[\sum_j \mathrm{Po}\left(cj\lambda_j(1-p_i)^{j-1}\right) < k-1\right] + o(1)\\ &= \Pr\left[\mathrm{Po}(c\lambda'(1-p_i)) < k-1\right] + o(1),\end{aligned}$$

and the argument proceeds as before. This demonstrates that as long as $c < c_k$, the peeling process applied to the double-hashing hypergraph leaves fewer than ϵm edges for any constant ϵ with high probability.

However, the final details of the argument are more delicate. Indeed, the statement that the k-core is empty with high probability in fact does not hold in the case of double hashing for 2-cores, although something close holds. To see this, consider the bottleneck of very small 2-cores consisting of two hyperedges that are the same. With random hyperedges of $j > 2$ vertices, the probability that two of the n edges are the same is at most $\binom{n}{2}\binom{m}{j}^{-1} = o(1)$, but in the case of double hashing the expected number of pairs of hyperedges that are the same is $\Omega\left(\binom{n}{2}\gamma_j^{-1}\right) = \Omega(1)$. Note that for k-cores with $k > 2$, in both settings the probability that k of the n hyperedges are the same is $o(1)$, although the probability is significantly smaller with random hyperedges as opposed to hyperedges from double hashing.

Our simulations demonstrate the difference regarding the 2-core does appear in practice; the difference, however, is between an empty core and one that in practice is roughly logarithmic in size. (There are heuristic arguments suggesting why with double hashing below the threshold the 2-core can be bounded by $O(\log m)$, but we do not present them here.) For some applications, this difference is unimportant; applications that require an empty 2-core, however, would need to find a suitable workaround.[1]

We conjecture that with the exception of the 2-core as noted above, the k-core threshold behavior is the same for double hashing, in that below the threshold with high probability the k-core is empty. Specifically, we make the following conjecture, which corresponds to a theorem proven for random hypergraphs in [22, Theorem 1] and gives a formula for the exact threshold at which a non-empty core appears. Let $D^d_{m,n}$ be a randomly chosen d-uniform double hashing random hypergraph with m nodes and n edges.

Conjecture 2: *Let $d, k > 2$. Define*

$$c^*_{k,d} = \min_{x>0} \frac{x}{d\left(1 - \mathrm{e}^{-x}\sum_{i=0}^{k-2}\frac{x^i}{i}\right)^{d-1}}.$$

1) *For any $c < c^*_{k,d}$, with probability $1 - o(1)$ the k-core of $D^d_{m,cm}$ is empty.*
2) *For any $c > c^*_{k,d}$, with probability $1 - o(1)$ the k-core of $D^d_{m,cm}$ has size $\alpha(c)n + o(n)$, where*

$$\alpha(c) = 1 - \mathrm{e}^{-x}\sum_{i=0}^{k-1}\frac{x^i}{i},$$

and x is the greatest solution to

$$c = \frac{x}{d\left(1 - \mathrm{e}^{-x}\sum_{i=0}^{k-2}\frac{x^i}{i}\right)^{d-1}}.$$

Further, for $d > 2$ and $k = 2$, the above also hold, except that when $c < c^*_{k,d}$, we have with probability $1 - o(1)$ the k-core of $D^d_{m,cm}$ is $O(\log n)$.

In the following, we prove Part 1 of Conjecture 2 for a limited range of parameters.

Theorem 3: *Let k, d be constants such that $k > d$. For all $c < c^*_{k,d}$, it holds that with probability $1 - o(1)$ the k-core of $D^d_{m,cm}$ is empty.*

Proof: The discussion above showed that if $c < c^*_{k,d}$, the peeling process applied to the double-hashing hypergraph leaves fewer than ϵm edges for any constant ϵ with high probability. So to complete the proof of the theorem, it suffices to show that with probability $1 - o(1)$, there is some $\epsilon' > 0$ such that $D^d_{m,cm}$ contains no subgraph with fewer than $\epsilon' m$ vertices and average degree at least k. Our proof follows the high-level outline of [22, Lemma 5].

As before, we find it slightly easier to analyze the graph $C^d_{m,cm}$ in which each of the $\binom{m}{2}$ hyperdges appears independently with probability $p = cm/\binom{m}{2}$. Asymptotically, the graphs $C^d_{m,cm}$ and $D^d_{m,cm}$ are equivalent up to $o(1)$ terms for the purposes of this analysis.

Consider a with $d \le a \le \epsilon' m$, for an ϵ' to be named later. If $C^d_{m,cm}$ contains a subgraph G on a vertices with average degree at least k, then G has at least $\frac{ka}{d}$ edges; we may delete edges to obtain a subgraph G' on a vertices with exactly $\frac{ka}{d}$ edges (for simplicity, we assume throughout that $\frac{ka}{d}$ is an integer). Let N denote the number of such subgraphs G' in $C^d_{m,cm}$. We prove an upper bound on the expected value of N. To this end, notice that

$$\mathbf{E}[N] \le \binom{m}{a}\binom{\binom{a}{2}}{\frac{ka}{d}} p^{\frac{ka}{d}}.$$

In the above, $\binom{m}{a}$ counts the number of ways to choose the set S of a nodes in the subgraph, $\binom{a}{2}$ is an upper bound on the number of length-d arithmetic sequences containing only nodes in S, and $p^{\frac{ka}{d}}$ is the probability that any fixed set of $\frac{ka}{d}$ edges all appear in $C^d_{m,cm}$. We then find

$$\begin{aligned}
\mathbf{E}[N] &\le \binom{m}{a}\binom{\binom{a}{2}}{\frac{ka}{d}} p^{\frac{ka}{d}} \\
&\le \left(\frac{em}{a}\right)^a \left(\frac{e\binom{a}{2}}{\frac{ka}{d}}\right)^{\frac{ka}{d}} \left(\frac{cm}{\binom{m}{2}}\right)^{\frac{ka}{d}} \\
&\le C_1^a\left(\frac{m}{a}\right)^a \cdot a^{\frac{ka}{d}} \cdot (1/m)^{\frac{ka}{d}} \\
&= C_1^a\left(\frac{a}{m}\right)^{(\frac{k}{d}-1)a},
\end{aligned}$$

where C_1 is a universal constant.

Since k and d are constants with $k > d$, the final term equals $C_1^a\left(\frac{a}{m}\right)^{\delta a}$ for some constant $\delta > 0$. If $a < \epsilon' m$ where $\epsilon' = 1/2C_1^{1/\delta}$, then the above expression is at most $\frac{1}{2^{\delta a}}$. Let $\ell := \frac{100\log m}{\delta}$. For $a > \ell$, this quantity is at most $1/m^{100}$.

Furthermore, notice that for $a \le \ell$, $C_1^a\left(\frac{a}{m}\right)^{\delta a} \le 1/m^{\Omega(1)}$. Thus, the expected number of subgraphs of size at most $\epsilon' m$ with $\frac{ka}{d}$ edges is at most

$$\mathbf{E}[N] \le \sum_{a=d}^{\ell}\frac{1}{m^{\Omega(1)}} + \sum_{a=\ell}^{\epsilon' m}\frac{1}{m^{100}} = \frac{1}{m^{\Omega(1)}}.$$

By Markov's inequality, with probability $1 - 1/m^{\Omega(1)}$, there is no such subgraph. ■

We remark that the bottleneck in preventing extending this argument to a larger range of parameters appears to be in showing that there are much fewer than $\binom{a}{2}$ length-d arithmetic sequences containing only nodes in S. Our use of $\binom{a}{2}$ appears to be a very pessimistic bound, particularly for small values of a. However, tightening this bound appears to require making some non-trivial use of the structure of collections of arithmetic sequences. Doing so may allow one to show that Part 1 of Conjecture 2 holds for a larger range of parameters.

[1]As an example, Biff codes [21] require an empty 2-core; however, a logarithmic sized 2-core could be handled by using a small amount of additional error-correction on the original data.

III. The Differential Equations Argument

Differential equations have also been used to analyze peeling processes (e.g. [11], [16], [18], [25], [26]), and in particular the k-core for random hypergraphs [4], [16]. The approach is generally utilized by analyzing the following process: choose a random vertex of degree less than k, delete it and its adjacent hyperedges, and continue.

We explain how this can be used to develop a family of differential equations. For convenience, we explain only for the 2-core, and for the case where all hyperedges have fixed degree d; the approach can be generalized.

As each hyperedge has degree d, if we start with n hyperedges, we have nd pairs (e, v) where e is a hyperedge and v is a vertex adjacent to that hyperedge. Let $X_i(0)$ denote the initial number of pairs where the vertex has degree i. Our algorithm, at each step, chooses a random vertex of degree 1, and deletes it and the rest of the associated hyperedges. We can write equations denoting the expected change of $X_i(T)$ at each step T, where T runs from 0 up to at most $n-1$. The key to making this argument hold over time is that if we start with a random hypergraph with a given degree distribution, as we continue this process, the result at each step will continue to be a random hypergraph constrained to have the corresponding degree distribution obtained after deleting the vertex and adjacent hyperedges (see, e.g., [4], [16], [22]). Because of this, at each step, it vector of values X_i form a Markov chain, and our analysis allows us to study the limiting behavior of this Markov chain.

Let $\Delta X_i(T)$ be the change in X_i over a time step. We drop the dependence on T from the notation when the meaning is clear. At each step we lose a vertex of degree 1, and $d-1$ other edge pairs. Each of those edge pairs contains a random vertex; the edges adjacent to that vertex have their degree lowered by 1. Let T denote the number of steps that have been performed. Then

$$\mathsf{E}[\Delta X_1] = (X_2 - X_1)\frac{d-1}{d(n-T)-1} - 1,$$

and for $i \geq 1$

$$\mathsf{E}[\Delta X_i] = (X_{i+1} - X_i)\frac{i(d-1)}{d(n-T)-1}.$$

(The probability that each of the $d-1$ edges hits a vertex of degree i is $\frac{X_i}{d(n-T)-1}$; when this happens, we lose i pairs from X_i, but increase X_{i-1} by $i-1$ pairs.)

In the limit as n grows to infinity, we can scale time so that $t = T/n$ runs from 0 to 1, and we can use $x_i(t) = X_i(tn)/(dn)$ to denote the correspondingly scaled fraction of the initial pairs that remain adjacent to vertices of degree i. Let $\Delta t = 1/n$. If we ignore the negligible -1 terms in the denominators, then the above equations can be written as

$$\frac{\mathsf{E}[\Delta x_1]}{\Delta t} = (x_2 - x_1)\frac{d-1}{d(1-t)} - \frac{1}{d},$$

and for $i \geq 1$

$$\frac{\mathsf{E}[\Delta x_i]}{\Delta t} = (x_{i+1} - x_i)\frac{i(d-1)}{d(1-t)}.$$

In the limit, these expectations will follow the corresponding differential equations

$$\frac{dx_1}{dt} = (x_2 - x_1)\frac{d-1}{d(1-t)} - \frac{1}{d},$$

and for $i \geq 1$

$$\frac{dx_i}{dt} = (x_{i+1} - x_i)\frac{i(d-1)}{d(1-t)}.$$

It can be shown that these differential equations in fact track the behavior of the algorithm for finding the 2-core. Heuristically, the differential equations can then be used to determine the size of the 2-core of a random hypergraph, including the threshold that separates empty from non-empty 2-cores. However, full formalizations of this can require additional work, much as in the case of the branching argument (see, e.g., [4]).

It is not immediately clear that the differential equations approach should apply in the double hashing setting. In particular, it is not clear at this point if the independence that one obtains in the fully random setting, where the hypergraph at each step can be taken to be a random graph (subject to the degree distribution), necessarily holds here. However, the recent argument in [20] shows that differential equations can be used for a load-balancing problem using choices from double hashing based on the near-independence of vertices, which follows from the fact that their local neighborhoods are disjoint with high probability. Extending such arguments to this domain remains an interesting open problem.

This framework suggests the following conjecture:

Conjecture 4: *The differential equations above describing the evolution of the randomized peeling algorithm for the 2-core of a random hypergraph also describe the evolution of the algorithm for the 2-core of a double hashing hypergraph. Similarly, the generalizations of these equations for the peeling algorithm for the k-core for random hypergraphs also apply to double hashing hypergraphs.*

IV. Cuckoo Hashing

We review cuckoo hashing and its connection to random hypergraphs; for more background, see [19]. In standard cuckoo hashing [23], keys are placed into a hash table with each bucket containing one item, and each key has two choices of bucket locations determined by random hash values. The key is placed in one of its two choices if either is empty; if neither is empty, the key is placed in its first choice, and the key there is moved to its other choice, and so on as needed. Eventually an empty space is found and all keys are placed, or a cycle is found and a failure occurs. One means of handling a failure is to put the key in a separate stash. The stash must be searched in its entirety when performing lookups into the table, but if the stash is constant-sized this is not problematic; in practice in hardware, if the stash is small it can often be implemented in very fast content addressable memory (CAM). With two choices, if the ratio to the number of keys to the number of buckets, known as the load, is less than $1/2$, all keys can be placed with high probability; using even a small stash can greatly reduce the failure probability [13]. If we think of the buckets as vertices and the keys as edges (corresponding to their choice of buckets), then the problem of assigning each key its own bucket can be viewed as a random graph problem, where each edge must be oriented to one of its adjacent vertices and each vertex can have at most one edge oriented to it.

A natural generalization of standard cuckoo hashing is to give each key $d \geq 3$ or more choices, in which case we now have a random hypergraph instead of a random graph, with keys again corresponding to hyperedges and vertices to buckets. We can again ask if there is a *load threshold* under which each key can have its own bucket with high probability. (For now, we ignore the question of what algorithm finds such an assignment of keys to buckets – offline it can be turned into a maximum matching problem – and focus on the load threshold.) These thresholds have been determined [5], [7], [8], [9], [14].

One natural approach to find the load threshold utilizes peeling arguments. If a bucket has only one adjacent key, we can have that key placed in the bucket, and then the key is removed from consideration. This corresponds exactly to peeling until reaching the 2-core in the corresponding random hypergraph. Obviously, if the 2-core is empty, then all the keys have been placed. However, unlike many other algorithmic settings, we do not need the 2-core to be empty for cuckoo hashing to succeed. For example, if we ended with a 2-core consisting of two edges (keys) mapping to the same set of d buckets (vertices), that would be fine; there is room for both keys. Only if $d+1$ keys collide with the exact same d buckets do we fail or force a key into the stash.

Let us call the density of a graph the ratio between the number of edges and the number of vertices. One way of describing the problem above, where $d+1$ edges exist on d vertices, is that the subgraph on those d vertices is too dense. It happens that this idea characterizes the load threshold. Above the load threshold, with high probability the 2-core will have density greater than 1, implying keys cannot be placed into buckets with each key having its own bucket. Below the load threshold, with high probability not only will the 2-core have density less than 1, but further every subgraph has density less than 1 [7].

The above ideas generalize naturally to buckets that can hold more than 1 key. The load threshold when buckets hold k keys corresponds to the point at which the $(k+1)$-core has density k [8].

Given our conjecture that a double hashing hypergraph yields essentially the same size k-core as a corresponding random hypergraph, we might expect cuckoo hashing to have the same load thresholds with double hashing as with random hashing. This suggests the following more general conjecture:

Definition 5: *We call $c_{d,b}$* a load threshold *for cuckoo hashing with fully random hash functions with $d \geq 3$ bucket choices per key and capacity $b \geq 1$ per bucket for some constants d and b if the following property is satisfied. Suppose there are n keys to be stored and m buckets and let $c := n/m$. As $m \to \infty$, if $c < c_{d,b}$ then cuckoo hashing succeeds with probability $1 - o(1)$, and if $c > c_{d,b}$ then cuckoo hashing fails with probability $1 - o(1)$.*

Conjecture 6: *Let $c_{d,b}$ be a load threshold for cuckoo hashing with fully random hash functions. Then $c_{d,b}$ is also a load threshold for cuckoo hashing using double hashing in place of fully random hash functions.*

We emphasize that the density analysis that shows that cuckoo hashing succeeds is rather complex, and while Conjecture 2 or Conjecture 4 would show that double hashing would yield the same size core as random hashing, this would not suffice to prove Conjecture 6, which would further require a generalization of the density analysis.

Finally, because with cuckoo hashing one can use a stash to handle a small (generally a constant) number of items, asymptotically small events (such as $d+1$ keys mapping the same d locations) can be ameliorated by use of a stash. We might therefore further expect that any differences between cuckoo hashing using double hashing and random hashing might be easily managed via, at worst, a slightly larger stash.

V. Simulations

A. Simulation Results for k-core

As we have noted previously, in the case of the 2-core, one would expect to see a difference in performance between random hashing and double hashing, and we see this difference in our simulations. As an example, consider hypergraphs with 4 vertices per hyperedge. Over 10000 simulations on random hypergraphs with 800000 hyperedges and 1048576 vertices, the 2-core size was always 0; these results are unsurprising as the edge density lies sufficiently below the threshold for an empty 2-core for random hypergraphs, which calculated numerically is approximately 0.77228. However, when using hypergraphs derived from double hashing, small 2-cores remain with non-trivial probability, as shown in Table I. The remaining cores are very small – the largest in these trials was 14 – but this demonstrates the difference in behavior. Indeed, even well before the threshold for an empty 2-core, we find that small 2-cores are likely to remain when using double hashing, as the simple expectation argument suggests. For example, Table II shows results for 700000 hyperedges.

On the other hand, above the threshold, for the 2-core the performance is quite similar. Over 10000 simulations on random hypergraphs with 820000 hyperedges and 1048576 vertices, the 2-core size was never 0 for both random and double hashing. The (min, mean, max) sizes of the 2-cores were (504180, 511503, 518191) for random hashing and (504640, 511458, 517930) for double hashing.

As mentioned previously, whether this difference in 2-core behavior is significant may depend on the underlying application. In some applications (e.g., [21]), an empty 2-core is expected, and in such cases the double hashing approach may be unsuitable or require a modification to handle the small 2-core that appears under double hashing.

Core	Random	Double
0	10000	3088
2	0	3715
4	0	2081
6	0	822
8	0	225
10	0	56
12	0	10
14	0	3

TABLE I
COMPARING RANDOM/DOUBLE 2-CORE SIZES, 800000 HYPEREDGES, 1048576 VERTICES, 4 VERTICES PER HYPEREDGE

Core	Random	Double
0	10000	4154
2	0	3612
4	0	1623
6	0	472
8	0	114
10	0	22
12	0	3

TABLE II
COMPARING RANDOM/DOUBLE 2-CORE SIZES, 700000 HYPEREDGES, 1048576 VERTICES, 4 VERTICES PER HYPEREDGE

As we shall see for cuckoo hashing with a stash [13], for some applications, the difference in behavior of the 2-core has no substantial effect.

Also as expected, this effect appears to vanish when looking at k-cores for $k > 2$. For example, in an experiment of 10000 simulations with 4 vertices per hyperedge, 340000 hyperedges, and 262144 vertices, the 3-core size was always 0 for both random and double hashing. (The load threshold for 3-cores with 4 vertices per edge is approximately 1.33364.) Moving above the load threshold, to 360000 hyperedges, the (min,mean,max) sizes of the 3-cores were (285129, 288296, 290952) for random hashing and (284565, 288308, 290969) for double hashing. Again, we see the nearly indistinguishable performance between double hashing and random hashing in this context.

B. Simulation Results for Cuckoo Hashing

Our simulations demonstrate that in fact there is little difference between d independent choices and d choices from double hashing in the cuckoo hashing setting. Simulations below show results using four choices, in conjunction with a stash; when a key is placed in the table, a key is put in the stash after 500 moves if the collision has not been resolved. The natural way to compare the two schemes is to examine the distribution of the size of the stash over many trials. We performed 1000000 trials for each of several settings, covering both where the load rarely requires a stash to where a stash is essentially always needed.

The following results are typical. With a hash table of size $2^{17} = 131072$ and 124000 keys, a stash is not needed for over 99% of the trials for both settings, as shown in Table III. While double hashing requires a stash of size 2 in slightly more trials, this still represent only 0.0042% of the trials, and the deviation between the performance of random and double hashing does not appear statistically significant. (In another set of 1000000 trials, we found the double hashing had fewer trials requiring a stash of size 2 than random.) Similarly, the distributions of the stash sizes for 125000 and 126000 keys, where the loads are over 95% but the stash size remains reasonable for many practical settings, are nearly the same, as shown in Tables IV and V. It does not appear that double hashing creates significant differences in stash behavior, even in somewhat overloaded tables.

Stash	Random	Double
0	992174	992004
1	7801	7970
2	25	42

TABLE III
COMPARING RANDOM/DOUBLE STASH SIZES, 124000 ITEMS, HASH TABLE SIZE 131072, LOAD 94.60%, 4 CHOICES

Stash	Random	Double
0	742167	742103
1	219439	219483
2	34306	34318
3	3718	3719
4	345	358
5	22	17
6	3	1
7	0	1

TABLE IV
COMPARING RANDOM/DOUBLE STASH SIZES, 125000 ITEMS, HASH TABLE SIZE 131072, LOAD 95.36%, 4 CHOICES

VI. CONCLUSION

Motivated in large part by a desire to simplify the implementation of several hash-based data structures and algorithms, we have argued for the study of certain families of hypergraphs generated via double hashing, and hence dubbed double hashing hypergraphs. We presented empirical results suggesting that in many algorithmic settings this subclass of random hypergraphs behaves essentially identically to their fully random counterparts. We also presented some partial theoretical justification for our empirical results. The main open question is to provide a more complete theoretical understanding of double hashing hypergraphs. In particular, a full proof of Conjectures 2, 4, and 6, which cover the behavior of double hashing hypergraphs with regard to the k-core and cuckoo hashing, appear to be natural next steps. While analysis for algorithms and data structures based on peeling arguments and the k-core served as our primary motivation, there may well be other uses for double hashing hypergraphs beyond these applications.

REFERENCES

[1] P. Bradford and M. Katehakis. A probabilistic study on combinatorial expanders and hashing. *SIAM Journal on Computing*, 37(1):83-111, 2007.

[2] A. Broder, A. Frieze, and E. Upfal. On the satisfiability and maximum satisfiability of random 3-CNF formulas. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 322–330, 1993.

Stash	Random	Double
0	1460	1416
1	8709	8791
2	26800	26662
3	56008	55640
4	90195	90372
5	118857	118647
6	135203	135158
7	134631	134405
8	120740	120710
9	98219	98296
10	73782	74441
11	51756	51970
12	34292	34081
13	21386	21386
14	12670	12638
15	7202	7271
16	3967	3945
17	2149	2113
18	1000	1057
19	536	503
20	229	264
21	117	134
22	46	54
23	29	27
24	9	12
25	5	3
26	3	4

TABLE V

COMPARING RANDOM/DOUBLE STASH SIZES, 126000 ITEMS, HASH TABLE SIZE 131072, LOAD 96.13%, 4 CHOICES

[3] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.

[4] R.W.R. Darling and J.R. Norris. Differential equation approximations for Markov chains. *Probability Surveys*, 5:37-79, 2008.

[5] M. Dietzfelbinger, A. Goerdt, M. Mitzenmacher, A. Montanari, R. Pagh, and M. Rink. Tight thresholds for cuckoo hashing via XORSAT. In *The 37th International Colloquium on Automata, Languages and Programming*, pp. 213-224, 2010.

[6] P.C. Dillinger and P. Manolios. Bloom Filters in Probabilistic Verification. In *Proceedings of the 5th International Conference on Formal Methods in Computer-Aided Design*, pp. 367-381, 2004.

[7] N. Fountoulakis and K. Panagiotou. Sharp load thresholds for cuckoo hashing. *Random Structures & Algorithms*, 41(3):306333, 2012.

[8] N. Fountoulakis, M. Khosla, and K. Panagiotou. The multiple-orientability thresholds for random hypergraphs. *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1222-1236, 2011.

[9] A. Frieze and P. Melsted. Maximum matchings in random bipartite graphs and the space utilization of cuckoo hash tables. Arxiv preprint arXiv:0910.5535, 2009.

[10] L. Guibas and E. Szemeredi. The analysis of double hashing. *Journal of Computer and System Sciences*, 16(2):226-274, 1978.

[11] R. Karp and M. Sipser. Maximum matching in sparse random graphs. In *Proceedings of the 22nd Annual Symposium on Foundations of Computer Science*, pp. 364–375, 1981.

[12] A. Kirsch and M. Mitzenmacher. Less hashing, same performance: Building a better Bloom filter. *Random Structures & Algorithms*, 33(2):187-218, 2008.

[13] A. Kirsch, M. Mitzenmacher, and U. Wieder. More robust hashing: cuckoo hashing with a stash. In *Proceedings of the* 16^{th} *Annual European Symposium on Algorithms*, pp. 611-622, 2008.

[14] L. LeLarge. A new approach to the orientation of random hypergraphs. *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 251-264, 2012.

[15] M. Luby, M. Mitzenmacher, and M. A. Shokrollahi. Analysis of random processes via and-or tree evaluation. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 364–373, 1998.

[16] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman. Efficient erasure correcting codes. *IEEE Transactions on Information Theory*, 47(2):569-584, 2001.

[17] G. Lueker and M. Molodowitch. More analysis of double hashing, *Combinatorica*, 13(1):83-96, 1993.

[18] M. Mitzenmacher. Tight thresholds for the pure literal rule. DEC SRC Technical Note 11, 1997.

[19] M. Mitzenmacher. Some open questions related to cuckoo hashing. In *Proceedings of the European Symposium on Algorithms*, pp. 1-10, 2009.

[20] M. Mitzenmacher. Balanced Allocations and Double Hashing. arXiv:1209.5360.

[21] M. Mitzenmacher and G. Varghese. Biff (Bloom filter) codes: Fast error correction for large data sets. In *Proceedings of the IEEE International Symposium on Information Theory*, pp. 483-487, 2012.

[22] M. Molloy. Cores in random hypergraphs and boolean formulas. In *Random Structures & Algorithms*, 27(1):124–135, 2005.

[23] A. Pagh and F. Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122-144, 2004.

[24] M. Pătraşcu and M. Thorup. The power of simple tabulation hashing. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*, pp.1-10, 2011.

[25] B. Pittel, J. Spencer, and N. Wormald. Sudden emergence of a giant k-core in a random graph. Journal of Combinatorial Theory Series B, 67(1):111-151, 1996.

[26] N.C. Wormald. Differential equations for random processes and random graphs. *The Annals of Applied Probability* 5 (1995) pp. 1217–1235.