Explicit, Closed-form, General bounds for Cuckoo Hashing with a Stash

Daniel Noble
University of Pennsylvania, dgnoble@cis.upenn.edu

June 16, 2023

Abstract

Cuckoo Hashing is a dictionary data structure in which a data item is stored in a small constant number of possible locations. It has the appealing property that a data structure of size $2m$ can hold up to $n = \frac{1}{d}m$ elements for any constant $d > 1$; *i.e.,* the data structure size is a small constant times larger than the combined size of all inserted data elements. However, the probability that a cuckoo hash table build fails is $\Theta(\frac{1}{m})$. This is too high for many applications, especially cryptographic applications and Oblivious RAM. An alternative proposal introduced by Kirsch et al. is to store elements which cannot be placed in the main table in a "stash", reducing the failure probability to $\mathcal{O}(m^{-(s+1)})$ where s is any constant stash size. However, this analysis did not apply to super-constant s, and the bounds are asymptotic rather than explicit. Further works improved upon this, but either were not explicit, not closed-form or had limitations on the stash size. In this paper we present the first explicit, closed-form bounds for the failure probability of cuckoo hashing with a stash for general stash sizes.

1 Overview

This paper proves the first explicit, closed-form bounds for the failure probability of cuckoo hashing with a stash for general stash sizes. Specifically, it proves the following bound:

Theorem 1. *Given a 2-table cuckoo hash table, in which each table holds m elements, and which has a stash of size $s \geq 0$, the cuckoo hash table can successfully hold $n \leq \frac{1}{d}m$ elements for any $d > 1$, with build failure*

probability at most:

$$C(s+2)\left(\frac{c(s+1)}{m}\right)^{s+1}$$

where C and c depend only on d:

$$C = \frac{16e^2 d}{(d-1)(\ln(d)+\frac{1}{d}-1)^2} e^{\frac{32d}{e(d-1)(\ln(d)+\frac{1}{d}-1)^2}}$$
$$c = \frac{1}{e(d-1)(\ln(d)+\frac{1}{d}-1)^2}$$

This bound has the following implications.

Negligible Failure for Small Tables
Our bound shows that to obtain failure negligible in some parameter N, it suffices to have $s = \Theta(\log(N))$ and $m = \omega(\log(N))$. Specifically Theorem 1 implies:

Corollary 1. *The probability of build failure for a 2-table cuckoo hash table of size $m = \omega(\log(N))$ and a stash of size $s = \Theta(\log(N))$ is negligible in N.*

The best previous analysis was by Kirsch et al., who only showed that this could be achieved when $m = \Omega(\log^7(N))$ [KMW09]. This result was used extensively in Hierarchical ORAMs (e.g. [GMOT12, LO13, AKL+20, KM19]), which wished to use Cuckoo Hashing with a stash for many hash tables of varying size, including "small" tables. While the constants were not explicit, if we set the constant to 1 and set N to 2^{40} this condition only holds for $m > 2^{37}$, which is far from small. By improving this result, we greatly increase the feasibility of Hierarchical ORAMs based on Cuckoo Hashing with a stash. Furthermore, many of these protocols needed separate, less efficient, hash tables for tables of sizes $o(\log^7(N))$. In some cases our analysis can avoid the need for these additional inefficient hash tables, simplifying Hierarchical ORAMs.

Flexible Concrete Parameter Choices
Our bound can be used to obtain concrete parameters for a desired security level. This has some advantages compared to choosing parameters based on simulation, as has been done for instance by [PSSZ15]. Firstly, and most importantly, it allows for parameter choices when the failure probability, $2^{-\sigma}$ is very small. Simulation here encounters a Catch-22: ideally the protocol never fails over the total number of times it is ever executed, say W, but

to show this would require running at least W executions in the simulation. It is rare for a protocol to be executed more times by researchers than in production. Secondly, it provides flexibility: it allows easy computation of bounds for the failure probability for new sets of parameters without needing to re-run expensive simulations. Lastly, on a philosophical level, it is more satisfying to base results on analysis than heuristics.

Table 1 below shows concrete parameter choices obtained from Theorem 1. It sets $d = 3$ and the failure probability to 2^{-40}. Compared to the simulation-based parameters of [PSSZ15], these parameters are a moderate constant times larger: the memory usage is 2.5x larger and the stash sizes about 4x larger. (See Table 4 of [PSSZ15]).

Number of Elements n	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}	2^{24}
Stash Size s	47	28	20	16	13	11	10	9	8

Table 1: Stash sizes needed to obtain failure probability below 2^{-40} when $d = 3$.

Excess of Random Bipartite Graphs

Our result can also be understood (and will be proven in) the language of random graphs. Specifically, we will make use of the standard representation of cuckoo hashing as a problem of bounding the excess of a random bipartite graph. We will prove the first explicit closed-form general upper bound for the excess of balanced bipartite graphs with randomly chosen edges:

Theorem 2. *Let G be a bipartite multigraph with parts L and R, where $|L| = |R| = m$, and $n = \frac{1}{d}m$ edges each chosen independently and uniformly at random from $L \times R$. Let $\boldsymbol{ex}(G)$ represent the* excess *of G, that is the minimum number of edges that need be removed such that every connected component has at most one cycle. Then:*

$$\boldsymbol{Pr}(\boldsymbol{ex}(G) \geq t) \leq C(t+1)\left(\frac{ct}{m}\right)^t$$

for every $t \geq 1$ and $d > 1$ where C and c are defined as above.

The paper is organized as follows. Section 2 presents a summary of previous work. Section 3 explains cuckoo hashing and Section 4 explains the correspondence between cuckoo hashing and random bipartite graphs. Section 5 presents the main analysis on the graph problem, though proofs for some of the lemmas for this analysis are deferred to Section 6. As a bonus,

Section 7 shows that Corollary 1 is basically tight; that is if $m = \mathcal{O}(\log(N))$ it is essentially impossible to attain negligible failure probability in N.

2 Previous Work

Cuckoo hashing, introduced by Pagh and Rodler [PR01], is a hash table implementation that improves performance by allowing objects to be stored in a number of locations. Specifically, a cuckoo hash table can store n elements of size W in $\Theta(nW)$ space, with accesses only accessing $\Theta(1)$ locations of size W. It is still sometimes impossible to place all items in the required locations, in which case the build fails. A build failure, which we henceforth refer to just as failure, occurs with probability $\Theta(\frac{1}{n})$ (see [DK12] for the explicit constant).

For many applications this failure probability is sufficient. If a failure occurs the hash table can simply be rebuilt with new hash functions. While a hash table rebuild requires $\Theta(n)$ computation, the probability of this occurrence is $\Theta(\frac{1}{n})$, so the *amortized* computation cost per access is still constant.

However, for many applications this failure probability is still too high. In particular, in security-oriented applications, rebuilds often constitute a security failure.

Kirsch, Mitzenmacher and Wieder therefore introduced the modification that any items which could not be stored in the main tables would be stored in a "stash" of constant size s [KMW09]. They showed that the probability of a build failure[1] was then reduced to $\mathcal{O}(n^{-(s+1)})$. (They state this as the equivalent statement that the probability that the required stash is at least s is $\mathcal{O}(n^{-s})$.) This analysis allowed the failure probability to be reduced significantly, but it only applied to constant s.

Kutzelnigg provided a detailed analysis of cuckoo hashing with a stash based on a generating function approach [Kut10]. He showed (Theorem 1) that the failure probability is exactly $c(\alpha, s)m^{-s-1} - \mathcal{O}(m^{-s-2})$ where m is the size of each table, s is the stash size, α is related to the ratio between the size of the table and the number of entries and c depends only on α and s. Note that $c(\alpha, s)$ is constant in m, but *is* a function of s. Therefore, this result does not show how the failure probability increases with s, even asymptotically. Furthermore, determining $c(\alpha, s)$ is extremely computationally intensive, effectively limiting this result to very small values of s. In particular: "The calculations [to compute $c(\alpha, s)$] are limited by

[1] While this paper presents failure probabilities in terms of m, if other papers present their results in terms of n, we do too. Since $n = \frac{1}{d}m$ converting between these is trivial.

the available memory of the machine that executed the computer algebra system. Using a workstation with 12GB RAM, we were successful in solving the problem for $s \in \{0, 1, 2\}$." [Kut10] Thus, it seems that a standard workstation is not able to compute $c(\alpha, s)$ for $s > 2$. Therefore, while this is a powerful result that demonstrates the asymptotic behavior in terms of m and provides precise bounds for small constant s, it does not provide upper bounds on the failure probability for general values of s.

Goodrich and Mitzenmacher developed another analysis for super-constant values of s in order to generate an improved Oblivious RAM protocol [GM11]. Oblivious RAM is a technique to hide virtual accesses from an adversary who can see physical accesses, and numerous protocols have used cuckoo hashing for this purpose (*e.g.,* [PR10], [GMOT12], [KLO12]). In these protocols, cuckoo hashing build failures constitute a security failure. Failure is desired to be negligible in some parameter N, but tables can be much smaller than N, for instance they may be polylogarithmic in N. Goodrich and Mitzenmacher extended the analysis of Kirsch et al. to achieve negligible failure probability in certain cases. They proved that, provided $n = \Omega(\log^7(N))$, the probability of a build failure is upper bounded by $n^{-\Omega(s)}$ for general values of s. Thus, a stash of size $s = \Theta(\log(N))$ would result in a failure probability negligible in N.

Aumüller, Dietzfelbinger and Woelfel then presented an elegant alternative analysis of cuckoo hashing with a stash based on graph counting [ADW14]. In this analysis, they showed a new result for super-constant stash sizes. They showed that for sufficiently large n, the failure probability is $\mathcal{O}(n^{-\frac{s}{2}})$ when $s \leq n^{\frac{1}{3\varrho}}$, for a suitable constant ϱ. However this constraint on the relationship between s and n proves restrictive in practice. They state "a rough estimate ... shows that $\varrho = 27$ suffices", but this would impose the restriction that $s \leq n^{\frac{1}{81}}$. Even for $s = 2$, this makes the bound inapplicable for practical values of m.

Pinkas et al. [PSWW18a] observed that an analysis by Wieder [Wie16] provided a bound on the stash size that does not make any assumptions about the stash size (see Appendix C of the full version [PSWW18b]). While Wieder's proof does not present explicit bounds in the final result, Pinkas et al. filled in the missing details to present a explicit, general bound. However, the bound they present is still not closed-form.[2]

[2]We could make the bound closed-form, such as by applying Lemma 7 from this paper. This would result in the following bound for the failure probability:

$$\left(\frac{2}{m}\right)^{s+1} 2e \left(\frac{8(s+1)+1}{e\ln(d)}\right)^{8(s+1)+1}$$

Pinkas et al. [PSSZ15] instead used simulation to calculate the failure probability of cuckoo hashing with a stash. Specifically, they generated random instances of the problem for $n \in \{2^{11}, 2^{12}, 2^{13}, 2^{14}\}$ and $d = 1.2$. For each value of n they generated 2^{30} random instances of the cuckoo hashing table and determined the stash sizes that were needed. From this they extrapolated the stash sizes needed for smaller error probabilities and other values of n (see [PSSZ15] Table 3 and Figure 1).

Two recent works investigate the interplay between stashes and other modifications to standard cuckoo hashing. Minaud and Papamanthou analyze a variant of cuckoo hashing with a stash in which each location has a capacity $\ell > 1$ [MP23]. They show that if ℓ is a (sufficiently large[3]) constant and the stash size s is also constant, the failure probability is upper-bounded by $\Theta(n^{-\ell-s})$ (Theorem 2). They also prove an asymptotic bound that applies to super-constant ℓ and s (Theorem 1), which again does not apply when $\ell = 1$. Yeo explores the effectiveness of increasing the number of hash functions k. He shows a *lower bound*[4] on the efficacy of combining all 3 parameters: that to achieve failure probability ϵ requires that $k^2\ell + ks = \Omega(\log(1/\epsilon)/\log(n)$. He also shows an upper bound when many hash functions are used and for constant s and ℓ: a failure probability ϵ can be achieved by using $k = \Theta(\sqrt{\log(1/\epsilon)/\log(n)})$ hash functions.

Most of these results did not provide explicit constants [KMW09, GM11, ADW14, Wie16]. Additionally, some of these restrict the stash size either to a constant ([KMW09]) or to a very small polynomial in n ([ADW14]). [Kut10] is also restricted to small stash sizes due to the difficulty of calculating the term $c(\alpha, s)$. [PSWW18a] is explicit and general, but is not closed form. [PSSZ15] is not directly comparable as it presents specific concrete parameter choices based on simulation. [MP23] and the upper bound of [Yeo22] are also not directly comparable as they either increase the number

For this probability to be less than 1 would require, roughly, that $m > (8(s+1))^8$. Even for $s = 2$ this would require that $m > 2^{36}$, a significant restriction in practice.

Asymptotically, the bound presented in Theorem 1 of this paper is much tighter. The bound above can be represented asymptotically as:

$$\mathcal{O}(\mathcal{O}(1)^s m^{-(s+1)} s^{8s+9})$$

whereas the equivalent representation of Theorem 1 would be:

$$\mathcal{O}(\mathcal{O}(1)^s m^{-(s+1)} s^{s+2})$$

[3]In particular, both theorems require that the capacity of each location is at least 5.5, that is at least $1 + \ln(1/\epsilon)/(1 - \ln(2))$ where $\epsilon \leq 0.25$.

[4]Theorem 2. See theorem statement for additional constraints.

of items per location or increase the number of hash functions. As such, despite much analysis of the build failure probability of cuckoo hashing with a stash, this is the first paper to present a concrete, explicit, closed-form upper bound.

3 Cuckoo Hashing

Cuckoo Hashing in its simplest form involves 2 hash functions, h_1 and h_2, and 2 hash tables, T_1 and T_2, each with $m = dn$ locations of capacity 1. Each hash table has a unique hash function, and the hash functions are assumed to produce outputs uniformly at random in $\{1, \ldots, m\}$. The tables consist of pairs (x, y) where x is the dictionary key and y is the dictionary value. An item (x, y) is stored in the table by being inserted into $T_1[h_1(x)]$. If another item (x', y') was stored in that location, it is removed from its original location (like a baby bird being displaced from its nest by a Cuckoo chick) and is placed in $T_2[h_2(x')]$. This may replace another item, which the algorithm likewise attempts to insert. This process continues either until every item has found a location in which to be inserted or it is determined that it is impossible to place all items in the cuckoo hash table.[5] In the latter case the insertion has "failed". This triggers a "table rebuild" in which new tables are created with new hash functions and the algorithm attempts to insert every element into the new hash table.

In cuckoo hashing with a stash, if an item cannot be inserted, the build does not immediately fail, but instead the item is placed in the stash. The build only fails if an item cannot be inserted and the stash is already full.

Cuckoo hash tables can be generalized to have a larger constant number of tables, or have locations with some constant capacity greater than 1. They can also be generalized to use multiple hash functions in a single table. However, this work will analyze only the traditional 2-table version.

[5] Some works fix a maximum recursion depth for the insertion procedure, such as [KMW09] which sets it to $\alpha \log(N)$ for a sufficiently large constant α. It is then possible, with a small but non-negligible (in N) probability, that an insertable item is not inserted. Our analysis instead assumes an optimal allocation. This can be achieved by only stopping the insertion when it is detected that the insertion process has entered an infinite loop. See Section 4 of [Kut10] for more details. This process still has an expected insertion time of $\Theta(1)$ per item (Lemma 3.7 of [Aum10]), so building the full table this way takes expected $\Theta(n)$ time.

4 Graph Representation

Analyses of cuckoo hash table failure often represent the problem as a graph problem as follows. For each location in the cuckoo hash table, create a vertex. Since the cuckoo hash table has two tables each of size m, there will be $2m$ vertices. For each element stored in the cuckoo hash table, draw an edge between the two locations in which it may be stored, so n edges total. Let G be the resulting graph. Since there will be one location from each table, G will be bipartite, with m vertices in each part. There may also be multiple edges between a pair of vertices, so G is a multigraph. Observe also that the graph is not connected: since $n < m$ some nodes will not be connected to any edges and there may also be multiple connected components that contain edges.

We introduce some graph notation and terminology. Given a graph G, let $\gamma(G)$ denote the cyclotomic number of G, that is the minimum number of edges that must be removed in order for G to have no cycles. Let $\mathbf{ex}(G)$ denote the *excess* of G, that is the minimum number of edges that must be removed from G to ensure that every connected component has at most one cycle.

Analysis is based on the following well-established observation (which is proven, for instance, as Lemma 5 of [ADW14]) which relates the build failure in a cuckoo hash table to the excess of its graph representation:

Theorem 3. *Let G be the graph representation of a cuckoo hash table with a stash of size s, where s is any non-negative integer. Then the build succeeds if and only if $\boldsymbol{ex}(G) \leq s$. Equivalently, the build fails if and only if $\boldsymbol{ex}(G) \geq s + 1$.*

Now, let $G(m, m, n)$ be the distribution of graphs generated from graph representations of random cuckoo hash tables with 2 tables of size m and n items. Let $G \leftarrow G(m, m, n)$, *i.e.,* G is randomly sampled from $G(m, m, n)$. As already stated, G will then be a bipartite graph with parts A and B each of m vertices. Since each hash function produces a random value in $\{1, \ldots, m\}$, G will have n edges chosen uniformly at random from $A \times B$. Note that this exactly matches the description of how G is chosen in Theorem 2. Therefore, Theorem 2 describes the bounds on the excess of graph representations of random cuckoo hash tables. Thus, by Theorem 3, Theorem 2 implies Theorem 1.

5 An Explicit Analysis

The remainder of this section proves Theorem 2, (though some more tedious lemmas are deferred to Section 6).

We follow the blueprint of Kirsch et al's analysis. [KMW09]. Their approach, in short, is to first bound the excess generated by each component of the graph. This is done by first bounding the size of each component, and calculating the distribution of the excess for a given size. Once they have bounds for the excess of each component, they sum this over the number of components to obtain a bound for the excess of the entire graph.

One core idea in Kirsch et al's analysis is to bound the excess of a related graph where the total number of edges is distributed according to a Poisson distribution $Po(\lambda)$, where $n < \lambda < m$. This means that the number of edges between any pair of edges is distributed according to *independent* Poisson distributions $Po(\frac{\lambda}{m^2})$. This has the benefit that revealing the number of edges between any pair of vertices will not affect the distribution between any other pair, much simplifying their analysis. However, it has the downside that there is a possibility that $Po(\lambda) < n$, that is the Poisson graph has fewer edges than the original graph. In this case the excess of the Poisson graph cannot be used to upper-bound that of the original. They parameterize the Poisson graph so that this occurs with failure probability $e^{-\Omega(n)}$. This term is added to the final failure probability. While this is asymptotically small, it complicates use of the bound in practice. To use the bound some concrete λ must be chosen: small λ increases the constant in the above $\Omega(\cdot)$ notation, large λ reduces the effective value of d in the remainder of the analysis.

We adopt a different approach. Instead of assuming that the existence of edges between pairs of vertices is independent, we carefully explore the graph in such a way such that, despite dependency, the distribution of edges between any pair will always be upper bounded by Binomial distributions. This is delicate. For instance, if we were to begin by picking a random vertex, v, in the left part of the graph and checking the number of neighbors it has, there is a chance that it has 0 neighbors. Conditioned on this event, the expected number of neighbors of any other left vertex increases, as there are still the same number of edges left to find but one fewer left vertex to use them. As such, our analysis begins by picking a random *edge* and carefully doing a Breadth First Search from this edge. Compared to [KMW09], this also allows us to sum component excesses starting from (at most) n edge-initiated Breadth First Searches rather than (at most) m vertex-initiated Breadth First Searches.

The algorithm below shows the (carefully chosen) method we use for

observing the contents of the graph. This algorithm will be used first to bound the size of components found, then the excess of each component. Finally this allows us to bound the excess of the entire graph.

Edge Component Search Algorithm

1. While $\exists$ an undiscovered edge in G
   (a) Select one such edge at random. Call it e. Call the vertices it connects v_1 and v_2.
   (b) Let Q_e be a queue initialized to $\{v_1, v_2\}$. Set $V_e = \{\}$, $Y_e = \{e\}$.
   (c) While Q_e is not empty
      i. Set $v \leftarrow \text{dequeue}(Q_e)$.
      ii. Add v to V_e.
      iii. Set N_v to be the set of undiscovered neighbors of v, (*i.e.,* neighbors of v which have never been placed in Q_e for any edge e). This should be thought of as first observing $|N_v|$, and then observing the vertices themselves.
      iv. Enqueue all vertices in N_v to Q_e.
      v. For each w in N_v, add one of the edges connecting v to w to Y_e. (If there is more that one such edge, pick one at random, without observing the total number of such edges.)
   (d) Set $T_e = (V_e, Y_e)$.
   (e) For every pair of vertices in V_e which are in different parts of G, observe the number of unobserved edges between the vertices. Set Z_e to be the set of these edges.
   (f) Set $C_e = (V_e, Y_e \cup Z_e)$.

Theorem 4. *C_e calculated in step 1f will be the connected component in G containing the edge e chosen in step 1a.*

Proof. Observe that steps 1b and 1c are identical to a Breadth First Search (BFS), except that the queue begins containing two vertices instead of 1. However, v_2 is a neighbor of v_1, so the initial state of the system can be viewed as the state of a BFS starting at v_1 where the first neighbor (v_2) has already been found, and added to the queue. Therefore, the resulting BFS will find exactly the nodes reachable from v_1 in G, which is exactly the nodes in the connected component in G containing e. Observe also that any edges that may exist in C_e are found, either in steps 1b and 1(c)v, in which

case they are added to Y_e, or in step 1e in which case they are added to Z_e. Either way, these edges exist in C_e. Lastly, only edges in the connected component containing e exist in C_e, since only edges in G connecting vertices in C_e are added. □

Observing the correspondence to a BFS also indicates the following Lemma.

Lemma 1. *T_e calculated in step 1d is a spanning tree of C_e.*

Furthermore, Y_e and Z_e are disjoint. Therefore, Z_e contains a set of edges in C_e which, if removed, produces a tree. This implies the following fact.

Fact 1. $|Z_e| = \gamma(C_e)$

Lastly, since the random edge selected in 1a will always be one that has not yet been discovered, and all edges in a component are discovered when that component is explored, each new component found will be separate to all previous components found. Furthermore, since the algorithm continues until all edges are found, it will find all components of G.

We will now prove some claims about the distibutions of neighbors, and edges, found by the Edge Component Search Algorithm. The analysis is made challenging by the fact that the existence of edges is not independent: the more edges are found, the fewer are left to find. Likewise, the more vertices are explored by the BFS, the fewer are left as possible end-points for the remaining edges. However this is resolved, in short, due to the way variables are observed in the Edge Component Search Algorithm, as edges are discovered at least as quickly as vertices are found in either part. Thus, even though the actual probability distributions depend on what has already been discovered, we can find probability distributions that only depend on m and n that stochastically dominate the real distributions.

We first upper-bound the number of neighbors found in step 1(c)iii.

Theorem 5. *The number of neighbors found in step 1(c)iii is stochastically dominated by Bin$(n, \frac{1}{m})$*

Proof. First, the number of undiscovered neighbors of v found in step 1(c)iii is at most the number of undiscovered edges that connect to v. Let u be the number of discovered edges at a certain point of time, and $n - u$ be the number of undiscovered edges. Let A be the part of the bipartite graph containing v and B the other part. Each of the u edges has one end-point

in A. One of these u edges is known to have its end-point in A at v (for v_1, v_2 this is e, and for other vertices, it is the edge that was used to find v). Therefore, there are $u-1$ edges that have other end-points in A, and so at most $u-1$ vertices in A that are end-points of previously discovered edges. Only vertices that are end-points of a previously-discovered edge may have their number of neighbors examined (in step 1(c)iii). Therefore, there are at least $m-u+1$ vertices in A (including v) which prior to to step 1(c)iii have not had their number of neighbors examined.

Some of the remaining $n-u$ edges may be later discovered to exist between previously found vertex-pairs (in step 1e). The number of undiscovered edges that are not in this category is still at most $n-u$.

Therefore, there are at most $n-u$ edges that could contribute towards $|N_v|$, and for each, the only thing that is known about the edge's end-point in A is that it is not one of the at most $u-1$ vertices in A which have had their number of neighbors counted. Hence, each such edge will have v as its end-point in A with probability at most $\frac{1}{m-u+1}$. Since there are at most $n-u$ such edges, $|N_v|$ is stochastically dominated by $\text{Bin}(n-u, \frac{1}{m-u+1})$, which by Lemma 3 is stochastically dominated by $\text{Bin}(n, \frac{1}{m})$

□

We now need to show bounds on the number of edges found in step 1e. It will help to first define three types of vertex pairs. The first are opened vertex pairs, for which the number of edges between the pair of vertices is fully known (including when it is known to be zero). Step 1e cannot find any edges between opened vertex pairs, since it only finds previously undiscovered edges. The second type is partially opened vertex pairs, for which it is known that at least one edge exists between them but it is not known how many more exist. The third type is unopened vertex pairs, for which it is not yet known whether the vertices are neighbors.

We begin by showing bounds on the number of edges between unopened vertex pairs.

Theorem 6. *In the Edge Component Search Algorithm, if at a point in time u edges have been discovered, then the number of edges between an unopened vertex pair, v and w, is stochastically dominated by Bin$(n, \frac{1}{m(m-u)})$.*

Proof. First we show that the number of edges between v and w is stochastically dominated by $\text{Bin}(n-u, \frac{1}{(m-u)^2})$.

Let q be the number of edges that exist between unopened vertex pairs. Every such edge must not yet have been discovered, but there may be some

undiscovered edges between partially opened vertex pairs. Therefore $q \leq n - u$

For the q edges that exist between unopened vertex pairs, we do not know any information about which vertices they exist between beyond the fact that they exist between unopened vertex pairs. Furthermore, it is equally likely to exist between any such pair.

Since only u edges have been discovered, there must be at least $m - u$ vertices in each part that touch no discovered edges. Hence, each pair of such vertices is an unopened vertex pair. Therefore, there are at least $(m-u)^2$ unopened vertex pairs. Thus, for any given unopened pair, and an edge that exists between a unopened pair, the probability that the edge exists between that unopened pair is at most $\left(\frac{1}{m-u}\right)^2$. Hence, the number of edges between any given unopened pair will be stochastically dominated by $\text{Bin}(n-u, \frac{1}{(m-u)^2})$, which by Lemma 4, is stochastically dominated by $\text{Bin}(n, \frac{1}{m(m-u)})$. □

Next we show bounds on the number of additional edges found in step 1e between partially opened pairs.

Theorem 7. *In the Edge Component Search Algorithm, the number of additional edges between a partially opened vertex pair is stochastically dominated by the number of edges between an unopened vertex pair.*

(Proof deferred to section 6).

Combining this with Theorem 6 and observing that $u \leq n$, we get the following result.

Theorem 8. *In the Edge Component Search Algorithm, any vertex pair that is partially opened or unopened, has a number of undiscovered edges that is stochastically dominated by Bin*$(n, \frac{1}{m(m-n)})$.

Now define a function $H(m, n)$, which samples a graph $H \leftarrow H(m, n)$ chosen the same as C_e in step 1 of the Edge Component Search Algorithm except that:

- Edges and vertices are given new unique identifiers when discovered that may not be the same as the names "found" by the Edge Component Search Algorithm.
- $|N_v|$ in step 1(c)iii is chosen from $\text{Bin}(n, \frac{1}{m})$

- In step 1e, the additional edges between any pair of vertices in different parts is chosen from $\text{Bin}(n, \frac{1}{m(m-n)})$. (Recall the graph is a tree at this point, so is bipartite.)
- We refer to V_e as V, Q_e as Q, Y_e as Y and Z_e as Z.

Theorem 9. *For any component C_e discovered in the Edge Component Search Algorithm, $\gamma(C_e)$ is stochastically dominated by $\gamma(H)$ for an independent sample $H \leftarrow H(m, n)$.*

Proof. We can view the two graph-sampling algorithms as running in parallel using the same source of randomness. We can choose an interpretation of the randomness generated such that if an event in the sampling of H stochastically dominates an event in the sampling of C_e, the event always happens in H if it happens in C_e. Since the probability of finding an edge in C_e is always stochastically dominated by that of finding the edge in H (from Theorems 5 and 8), C_e will be a subset of H for any choice of randomness. Therefore $\gamma(C_e) \leq \gamma(H)$ for any choice of randomness, which implies that $\gamma(C_e)$ is stochastically dominated by $\gamma(H)$. □

We can now upper bound $|H|$ and $\gamma(H)$ in order to upper bound $\gamma(C_e)$.

Theorem 10. *For $H \leftarrow H(m, n)$, where $m = dn$ for $d > 1$, and $c_1 = \ln(d) + \frac{1}{d} - 1$, for $k \geq 2$,*

$$\boldsymbol{Pr}(|H| \geq k) \leq \frac{2d^2}{k-1} e^{-c_1 k}$$

Proof. Now, the vertices of H are found by each vertex having a number of children chosen from the distribution $\text{Bin}(n, \frac{1}{m})$. Therefore (V, Y) can be viewed as the result of a Galton-Watson Branching process, with 2 roots, and children chosen from independent samples of $\text{Bin}(n, \frac{1}{m})$. The Otter-Dwass formula [Pit98, Dwa69] states that the probability that a Galton-Watson process that initially has α nodes, will be of size k is exactly

$$\frac{\alpha}{k} \mathbf{Pr}(S_k = k - \alpha)$$

where S_k is the distribution of k samples of the progeny distribution. In this case $S_k = \text{Bin}(nk, \frac{1}{m})$. Therefore, for $k \geq 2$

$$\mathbf{Pr}(|H| = k) = \frac{2}{k} \mathbf{Pr}(\text{Bin}(nk, 1/m) = k - 2)$$

We can rearrange the binomial to make it amenable to Chernoff Bounds:

$$\begin{aligned}\Pr(\text{Bin}\,(nk,1/m)=k-2) &= \frac{(kn)!}{(k-2)!(kn-k+2)!}\left(\frac{1}{m}\right)^{k-2}\left(1-\frac{1}{m}\right)^{kn-(k-2)}\\ &= \frac{k(k-1)}{(kn-k+2)(kn-k+1)}m^2\left(1-\frac{1}{m}\right)^{2}\frac{(kn)!}{k!(kn-k)!}\left(\frac{1}{m}\right)^{k}\left(1-\frac{1}{m}\right)^{kn-k}\\ &= \frac{k(k-1)}{(kn-k+2)(kn-k+1)}m^2\left(1-\frac{1}{m}\right)^{2}\Pr(\text{Bin}(nk,\frac{1}{m}=k)\end{aligned}$$

We then substitute this term and apply basic inequalities to simplify:

$$\begin{aligned}\mathbf{Pr}(|H|=k) &= \frac{2(k-1)}{(kn-k+2)(kn-k+1)}m^2\left(1-\frac{1}{m}\right)^{2}\mathbf{Pr}(\text{Bin}(nk,\frac{1}{m})=k)\\ &\leq \frac{2(k-1)}{(kn-k)(kn-k)}m^2\mathbf{Pr}(\text{Bin}(nk,\frac{1}{m})=k)\\ &\leq \frac{2d^2(k-1)}{(k-\frac{k}{n})(k-\frac{k}{n})}\mathbf{Pr}(\text{Bin}(nk,\frac{1}{m})=k)\\ &\leq \frac{2d^2}{k-1}\mathbf{Pr}(\text{Bin}(nk,\frac{1}{m})=k)\\ &\leq \frac{2d^2}{k-1}\mathbf{Pr}(\text{Bin}(nk,\frac{1}{m})\geq k)\end{aligned}$$

A standard Chernoff bound can now be applied on the binomial distribution, where the expected value is $\mu = \frac{nk}{m} = \frac{k}{d}$ and the lower limit is $k = d\mu$.

$$\begin{aligned}\Pr(X\geq d\mu) &\leq \left(\frac{e}{d}\right)^{d\mu}e^{-\mu}\\ &\leq e^{k}e^{-k\ln(d)}e^{-\frac{k}{d}}\\ &\leq e^{-k(\ln(d)+\frac{1}{d}-1)}\end{aligned}$$

Substituting yields our final result, with $c_1 = \ln(d) + \frac{1}{d} - 1$:

$$\mathbf{Pr}(|H|=k)\leq\frac{2d^2}{k-1}e^{-c_1k}$$

□

Now we can bound $\gamma(H)$ for a given $|H|$.

Theorem 11. *For $H \leftarrow H(m,n)$,*

$$\boldsymbol{Pr}(\gamma(H) \geq t||H| = k) \leq \left(\frac{enk^2}{4m(m-n)t}\right)^t$$

Proof. H is bipartite. If one part has size a, the other has size $k-a$. The cyclotomic number of H is the number of additional edges added in the last step. The number of pairs of vertices that may have edges added between them is $a(k-a)$ which has maximum value $\lfloor \frac{k}{2} \rfloor \lceil \frac{k}{2} \rceil = \lfloor \frac{k^2}{4} \rfloor$.

Each such vertex pair has a number of edges drawn from the distribution $\text{Bin}(n, \frac{1}{(m-n)m})$. Therefore the total number of edges is stochastically dominated by $\text{Bin}(n\lfloor \frac{k^2}{4} \rfloor, \frac{1}{m(m-n)})$.

Applying the Chernoff bound from Lemma 6 completes the proof. □

Theorem 12. *For any component C_e found by the Edge Component Search Algorithm,*

$$\boldsymbol{Pr}(\gamma(C_e) \geq t) \leq c_2 \left(\frac{c_3 t}{m}\right)^t$$

where $c_2 = 8ed^2$ and $c_3 = \frac{1}{e(d-1)c_1^2} = \frac{1}{e(d-1)(\ln(d)+\frac{1}{d}-1)^2}$.

Proof. First we show bounds on $\gamma(H)$, where $H \leftarrow H(m,n)$. Combining Theorem 10 and Theorem 11 we can obtain bounds for $\gamma(H)$:

$$\begin{aligned}
\mathbf{Pr}(\gamma(H) \geq t) &\leq \sum_{k=2}^{\infty} \mathbf{Pr}(|H| = k)\mathbf{Pr}(\gamma(H) \geq t||H| = k) \\
&\leq \sum_{k=2}^{\infty} \frac{2d^2}{k-1} e^{-c_1 k} \left(\frac{enk^2}{4m(m-n)t}\right)^t \\
&\leq 2d^2 \left(\frac{en}{4m(m-n)t}\right)^t \sum_{k=2}^{\infty} \frac{1}{k-1} e^{-c_1 k} k^{2t} \\
&\leq 4d^2 \left(\frac{en}{4m(m-n)t}\right)^t \sum_{k=2}^{\infty} e^{-c_1 k} k^{2t-1}
\end{aligned}$$

Applying Lemma 7 yields:

$$\begin{aligned}\mathbf{Pr}(\gamma(H) \geq t) &\leq 4d^2 \left(\frac{en}{4m(m-n)t}\right)^t 2e \left(\frac{2t}{c_1 e}\right)^{2t} \\ &\leq 8ed^2 \left(\frac{t}{em(d-1)c_1^2}\right)^t\end{aligned}$$

Since $\gamma(C_e)$ is stochastically dominated by $\gamma(H)$,

$$\mathbf{Pr}(\gamma(C_e) \geq t) \leq 8ed^2 \left(\frac{t}{e(d-1)c_1^2 m}\right)^t$$

□

This immediately implies the following corollary:

Corollary 2. *For any component C_e found by the Edge Component Search Algorithm,*

$$\boldsymbol{Pr}(\boldsymbol{ex}(C_e) \geq s) \leq c_2 \left(\frac{c_3(s+1)}{m}\right)^{s+1}$$

where $c_2 = 8ed^2$ and $c_3 = \frac{1}{e(d-1)c_1^2} = \frac{1}{e(d-1)(\ln(d)+\frac{1}{d}-1)^2}$.

Note that this bound not only applies to the first component found, but to *every* component found.

Let C'_e be the component containing e if e is the first edge found in the Edge Component Search Algorithm and let C'_e be an empty component otherwise. $\mathbf{ex}(C'_e) = \mathbf{ex}(C_e)$ if e is the first edge found in C_e, and $\mathbf{ex}(C'_e) = 0$ otherwise. In either case the bound of Corollary 2 applies to $\mathbf{ex}(C'_e)$.

We will need the following Lemma (proven in section 6).

Lemma 9. *Let $U(s,q)$ be the set of sequences of positive integers, where $T \in U$ if and only if $|T| = q$ and $\sum_{1 \leq i \leq q} T_i = s$, where $s \geq q \geq 1$. Then $\sum_{T \in U(s,q)} \prod_{1 \leq i \leq q} (T_i + 1)^{T_i+1} \leq \left(\frac{4}{e}\right)^{q-1} (s+1)^{s+1}$*

We can now bound the excess of the entire graph,

$$\mathbf{Pr}(\mathbf{ex}(G) \geq s) = \mathbf{Pr}(\sum_e \mathbf{ex}(C'_e) \geq s)$$

$$
\begin{aligned}
&\leq \sum_{\substack{j_1,\ldots,j_n \\ \sum_i j_i = s}} \mathbf{Pr}(\wedge_i \mathbf{ex}(C'_{e_i}) \geq j_i) \\
&\leq \sum_{\substack{j_1,\ldots,j_n \\ \sum_i j_i = s}} \prod_{\{j_i : j_i \geq 1\}} c_2 \left(\frac{c_3(j_i+1)}{m}\right)^{j_i+1} \\
&\leq \sum_{q=1}^{s} \sum_{\substack{j_1,\ldots,j_n \\ \sum_i j_i = s \\ |\{j_i : j_i \geq 1\}| = q}} \prod_{\{j_i : j_i \geq 1\}} c_2 \left(\frac{c_3(j_i+1)}{m}\right)^{j_i+1} \\
&\leq \sum_{q=1}^{s} \sum_{\substack{R \subseteq \{1,\ldots,n\} \\ |R| = q}} \sum_{\substack{j_1,\ldots,j_q \\ \sum_i j_i = s \\ j_i \geq 1}} \prod_{i=1}^{q} c_2 \left(\frac{c_3(j_i+1)}{m}\right)^{j_i+1} \\
&\leq \sum_{q=1}^{s} \binom{n}{q} \sum_{T \in U(s,q)} \prod_{i=1}^{q} c_2 \left(\frac{c_3(T_i+1)}{m}\right)^{T_i+1} \\
&\leq \sum_{q=1}^{s} \binom{n}{q} c_2^q \left(\frac{c_3}{m}\right)^{s+q} \sum_{T \in U(s,q)} \prod_{i=1}^{q} (T_i+1)^{T_i+1}
\end{aligned}
$$

We then apply Lemma 9, binomial approximation $\binom{n}{q} \leq \left(\frac{en}{q}\right)^q$ and inequality $(s+1)^{s+1} = (s+1)s^s((s+1)/s)^s = (s+1)s^s(1+1/s)^s \leq (s+1)s^s e$.

$$
\begin{aligned}
\mathbf{Pr}(\mathbf{ex}(G) \geq s) &\leq \sum_{q=1}^{s} \binom{n}{q} c_2^q \left(\frac{c_3}{m}\right)^{s+q} \left(\frac{4}{e}\right)^{q-1} (s+1)^{s+1} \\
&\leq \sum_{q=1}^{s} \left(\frac{en}{q}\right)^q c_2^q \left(\frac{c_3}{m}\right)^{s+q} \left(\frac{4}{e}\right)^{q-1} (s+1)^{s+1} \\
&\leq \frac{1}{\left(\frac{4}{e}\right)} (s+1)^{s+1} \left(\frac{c_3}{m}\right)^s \sum_{q=1}^{s} \left(\frac{\left(\frac{4}{e}\right) e c_2 c_3}{dq}\right)^q \\
&\leq \frac{(s+1)e}{\left(\frac{4}{e}\right)} \left(\frac{c_3 s}{m}\right)^s \sum_{q=1}^{s} \left(\frac{\left(\frac{4}{e}\right) e c_2 c_3}{dq}\right)^q
\end{aligned}
$$

Finally, we apply Lemma 10:

$$
\mathbf{Pr}(\mathbf{ex}(G) \geq s) \leq \frac{(s+1)e}{\left(\frac{4}{e}\right)} \left(\frac{c_3 s}{m}\right)^s \frac{2e\left(\frac{4}{e}\right) c_2 c_3}{d} e^{\frac{\left(\frac{4}{e}\right) c_2 c_3}{d}}
$$

$$\leq (s+1)\frac{2e^2c_2c_3}{d}e^{\frac{4c_2c_3}{ed}}\left(\frac{c_3 s}{m}\right)^s$$

Setting our constants accordingly completes the proof of Theorem 2:

$$C = \frac{2e^2c_2c_3}{d}e^{\frac{4c_2c_3}{ed}} = \frac{16e^2 d}{(d-1)(\ln(d)+\frac{1}{d}-1)^2}e^{\frac{32d}{e(d-1)(\ln(d)+\frac{1}{d}-1)^2}}$$

$$c = c_3 = \frac{1}{e(d-1)(\ln(d)+\frac{1}{d}-1)^2}$$

6 Additional Lemmas

This section contains the deferred proofs of useful lemmas. We begin by proving an inequality which will be useful in showing stochastic dominance between Binomial distributions.

Lemma 2.

$$ax \leq by, a, b, y > 0, x > 1 \Rightarrow \left(1-\frac{1}{y+1}\right)^a \geq \left(1-\frac{1}{x}\right)^b$$

Proof. Under the above parameters, $ax \leq by \Rightarrow \frac{a}{y} \leq \frac{b}{x} \Rightarrow -\frac{a}{y} \geq -\frac{b}{x}$.

$$-\frac{a}{y} = a\left(-\frac{1}{y}\right) = a\left(1-\frac{y+1}{y}\right) = a\left(1-\frac{1}{y/(y+1)}\right)$$

$\frac{1}{y/(y+1)} > 0$ so we can use inequality $1-\frac{1}{s+1} \leq \ln(s+1)$ with $s+1 = y/(y+1)$.

$$-\frac{a}{y} \leq a\ln(\frac{y}{y+1}) = a\ln(1-\frac{1}{y+1})$$

Meanwhile $-\frac{b}{x} = b\left(-\frac{1}{x}\right)$. Since $x > 1$, $0 < \frac{1}{x} < 1$ so $-\frac{1}{x} > -1$. Therefore we can use inequality $s \geq \ln(s+1)$ with $s = -\frac{1}{x}$. Therefore:

$$-\frac{b}{x} \geq b\ln(1-\frac{1}{x})$$

$$a\ln(1-\frac{1}{y+1}) \geq b\ln(1-\frac{1}{x})$$

$$\left(1-\frac{1}{y+1}\right)^a \geq \left(1-\frac{1}{x}\right)^b$$

□

Lemma 3. *Bin*$(n-q, \frac{1}{m-q+1})$ *is stochastically dominated by Bin*$(n, \frac{1}{m})$ *when* $m \geq n > q \geq 0$.

Proof. Now $\mathrm{Bin}(n_1, p_1)$ is stochastically dominated by $\mathrm{Bin}(n_2, p_2)$ if and only if $n_1 \leq n_2$ and $(1-p_1)^{n_1} \geq (1-p_2)^{n_2}$ [KM10]. Clearly $n - q \leq n$, so the first condition is satisfied. The second condition will be satisfied if:

$$\left(1 - \frac{1}{m-q+1}\right)^{n-q} \geq \left(1 - \frac{1}{m}\right)^{n} \tag{1}$$

Now $m \geq n > q \geq 0$, so:

$$\begin{aligned} qm &\geq qn \\ mn - qm &\leq mn - qn \\ m(n-q) &\leq n(m-q) \end{aligned}$$

If we apply Lemma 2 with $a = n - q$, $x = m$, $b = n$ and $y = m - q$, (note that all range requirements are satisfied) we obtain inequality (1). Hence the second condition is satisfied, completing the proof. □

Lemma 4. *If integers* m, n, u *satisfy* $m > n \geq u \geq 1$ *then Bin*$(n-u, \frac{1}{(m-u)^2})$ *is stochastically dominated by Bin*$(n, \frac{1}{m(m-u)})$.

Proof. $\mathrm{Bin}(n_1, p_1)$ is stochastically dominated by $\mathrm{Bin}(n_2, p_2)$ if and only if $n_1 \leq n_2$ and $(1-p_1)^{n_1} \geq (1-p_2)_2^n$ [KM10]. Clearly, $n - u \leq n$, so the first condition is satisfied. The second condition will be satisfied if:

$$\left(1 - \frac{1}{(m-u)^2}\right)^{n-u} \geq \left(1 - \frac{1}{m(m-u)}\right)^{n} \tag{2}$$

Since $m > n > u \geq 1$, $u(m-u)$ is minimized when $u = 1$ where it equals $m - 1$. It follows that:

$$\begin{aligned} u(m-u)(m-n) &\geq (m-1)(m-n) \geq m-1 \\ u(m-u)(m-n) &\geq n \\ (m-u)um &\geq (m-u)un + n \\ (m-u)(um-mn) &\geq (m-u)(un-mn) + n \\ (m-u)(mn-um) &\leq (m-u)(mn-un) - n \\ m(m-u)(n-u) &\leq n((m-u)^2 - 1) \end{aligned}$$

Applying Lemma 2 with $a = n - u$, $x = m(m-u)$, $b = n$ and $y = (m-u)^2 - 1$, (and noting all range requirements are satisfied), we see that Inequality (2) holds, completing the proof. □

Theorem 7. *In the Edge Component Search Algorithm, the number of additional edges between a partially opened vertex pair is stochastically dominated by the number of edges between an unopened vertex pair.*

Proof. The case of the pair v_1, v_2 is special because the initial edge e that was found between these was discovered by selecting a random undiscovered edge, rather that requesting information about the pair v_1, v_2. Therefore, the occurrence of e between v_1 and v_2 does not affect the distribution of other edges. Hence the remaining edges between v_1 and v_2 will actually be distributed exactly the same as between any unopened vertex pair.

For the remaining partially opened vertex pairs, we will prove a slightly different statement, which implies the one above. Let A be the number of additional edges between the partially opened pair and B be the number between the unopened pair. We show that if $i < j$, the probability that $A = i$ and $B = j$ is less than the probability that $A = j$ and $B = i$.

Let there be some state, S, observed on the remainder of the system. By Bayes:

$$\frac{\mathbf{Pr}(B = i \wedge A = j+1|S)}{\mathbf{Pr}(B = j \wedge A = i+1|S)} = \frac{\mathbf{Pr}(B = i \wedge A = j+1)\mathbf{Pr}(S|B = i \wedge A = j+1)}{\mathbf{Pr}(B = j \wedge A = i+1)\mathbf{Pr}(S|B = j \wedge A = i+1)}$$

Now $\mathbf{Pr}(S|B = i \wedge A = j+1) = \mathbf{Pr}(S|B = j \wedge A = i+1)$, since in both cases $i + j + 1$ edges will have been used between the two vertex-pairs. Therefore, the probability above is simply

$$\frac{\mathbf{Pr}(B = i \wedge A = j+1)}{\mathbf{Pr}(B = j \wedge A = i+1)}$$

Note that this statement is true regardless of what the state S is (as long as it is possible), so we can consider S to be all information learned about the assignment of edges from the beginning of the Edge Component Search Algorithm.

So, if there are initially b vertex pairs and n edges:

$$\frac{\mathbf{Pr}(B = i \wedge A = j+1|S)}{\mathbf{Pr}(B = j \wedge A = i+1|S)} = \frac{\mathbf{Pr}(B = i \wedge A = j+1)}{\mathbf{Pr}(B = j \wedge A = i+1)}$$

$$
\begin{aligned}
&= \frac{\mathbf{Pr}(B = i | A = j+1)\mathbf{Pr}(A = j+1)}{\mathbf{Pr}(B = j | A = i+1)\mathbf{Pr}(A = i+1)} \\
&= \frac{\binom{n-j-1}{i}\left(\frac{1}{b-1}\right)^{i}\left(1-\frac{1}{b-1}\right)^{n-j-1-i}\binom{n}{j+1}\left(\frac{1}{b}\right)^{j+1}\left(1-\frac{1}{b}\right)^{n-j-1}}{\binom{n-i-1}{j}\left(\frac{1}{b-1}\right)^{j}\left(1-\frac{1}{b-1}\right)^{n-i-1-j}\binom{n}{i+1}\left(\frac{1}{b}\right)^{i+1}\left(1-\frac{1}{b}\right)^{n-i-1}} \\
&= \frac{\binom{n-j-1}{i}\binom{n}{j+1}}{\binom{n-i-1}{j}\binom{n}{i+1}} \\
&= \frac{i+1}{j+1} < 1
\end{aligned}
$$

Therefore, $\mathbf{Pr}(B = i \wedge A = j+1|S) < \mathbf{Pr}(B = j \wedge A = i+1|S)$, when $i < j$, which implies that after observation of the system S, $A - 1$ is stochastically dominated by B. Therefore, the number of additional edges between a partially opened pair is stochastically dominated by the number of edges between an unopened pair. □

Lemma 5. *Let X be the sum of independent Bernoulli variables, with mean μ. A basic form of the Chernoff Bound for any $\delta > 0$ is as follows:*

$$\boldsymbol{Pr}(X \geq (1+\delta)\mu) \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

This implies the following looser bound:

Lemma 6. *For any non-negative integer t,*

$$\boldsymbol{Pr}(X \geq t) \leq \left(\frac{e\mu}{t}\right)^{t}$$

Proof. For $t \leq \mu$, $\frac{e\mu}{t} \geq e$, so $\left(\frac{e\mu}{t}\right)^{t} \geq 1$, so the statement holds as the probability cannot be more than 1. For $t > \mu$ we can view $t = (1+\delta)\mu$ for some $\delta > 0$.

$$
\begin{aligned}
\mathbf{Pr}(X \geq (1+\delta)\mu) &\leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu} \\
&\leq \left(\frac{e}{(1+\delta)}\right)^{(1+\delta)\mu} e^{-\mu} \\
&\leq \left(\frac{e\mu}{(1+\delta)\mu}\right)^{(1+\delta)\mu} e^{-\mu}
\end{aligned}
$$

$$\leq \left(\frac{e\mu}{t}\right)^t$$

□

Lemma 7. *Let $a \geq 1$, $b \in (0,1)$. Then:*

$$\sum_{k=1}^{\infty} k^a e^{-bk} \leq 2e \left(\frac{a+1}{be}\right)^{a+1}$$

Proof. It is well-known that a Riemann sum can be used to approximate an integral. Furthermore, if the summands are chosen to be the minimum point, the Riemann sum will lower-bound the integral:

$$\sum_{i=u}^{v-1} \min(f(i), f(i+1)) \leq \int_u^v f(x)dx$$

where u and v are integers.

It naturally follows that an integral can be used to upper-bound a summation. However, since we are taking the minimum of each adjacent pair, certain summands may be excluded, while others may be double-counted. Let us analyze the case where over the range $[u, v]$ $f(x)$ is continuous and monotonically increasing up to some maximum x_{max} after which it is monotonically decreasing, where $u < x_{max} < v$.

If x_{max} is an integer, then $f(x_{max})$ will not be included in the above sum, whereas for all other integers $i \in [u, v]$, $f(i)$ will be included. If x_{max} is not an integer, either $f(\lfloor x_{max} \rfloor)$ or $f(\lceil x_{max} \rceil)$ will not be included, whichever one is larger, and $f(i)$ will be included for all other integers $i \in [u, v]$. Either way, since x_{max} is the maximum on the curve $f(\cdot)$, the excluded element is still at most $f(x_{max})$. Therefore:

$$\sum_{i=u}^{v} f(i) \leq \sum_{i=u}^{v-1} \min(f(i), f(i+1)) + f(x_{max}) \leq \int_u^v f(x)dx + f(x_{max})$$

We can now apply this to the function in question. Let $f(x) = x^a e^{-bx}$ where $a \geq 1$ and $0 < b < 1$. First we need to show that $f(\cdot)$ is monotonically increasing, then monotonically decreasing.

$$f'(x) = ax^{a-1}e^{-bx} - bx^a e^{-b} = x^{a-1}e^{-b}(a - bx)$$

Observe that x^{a-1} and e^{-b} are both positive. Therefore $f'(x)$ will be positive when $x < \frac{a}{b}$, $f'(x) = 0$ at $x = \frac{a}{b}$ and will be negative when $x > \frac{a}{b}$. Therefore this function is monotonically increasing, then monotonically decreasing, as required, with $x_{max} = \frac{a}{b}$. We can easily calculate:

$$f(x_{max}) = \left(\frac{a}{b}\right)^a e^{-a}$$

Hence the inequality applies to the sum and

$$\begin{aligned}\sum_{k=1}^{\infty} k^a e^{-bk} &\leq \int_1^{\infty} x^a e^{-bx} dx + \left(\frac{a}{be}\right)^a \\ &\leq \int_0^{\infty} x^a e^{-bx} dx + \left(\frac{a}{be}\right)^a\end{aligned}$$

By a standard integral identity, $\int_0^{\infty} x^a e^{-bx} dx = \frac{a!}{b^{a+1}}$. Furthermore, a factorial approximation shows that $a! \leq (a+1)^{a+1} e^{-a}$. Hence:

$$\sum_{k=1}^{\infty} k^a e^{-bk} \leq \left(\frac{a+1}{b}\right)^{a+1} e^{-a} + \left(\frac{a}{b}\right)^a e^{-a}$$

Recall that $0 < b < 1$, so $\frac{a+1}{b} > 1$. Therefore:

$$\begin{aligned}\sum_{k=1}^{\infty} k^a e^{-bk} &\leq 2\left(\frac{a+1}{b}\right)^{a+1} e^{-a} \\ &\leq 2e\left(\frac{a+1}{be}\right)^{a+1}\end{aligned}$$

□

Lemma 8. *For all integers $s \geq 2$, $\sum_{a=1}^{s-1} (a+1)^{a+1}(s+1-a)^{s+1-a} \leq \left(\frac{4}{e}\right)(s+1)^{s+1}$*

Proof. Let $T(a) = (a+1)^{a+1}(s-a+1)^{s-a+1}$.

For $2 \leq a \leq \frac{s}{2}$

$$\frac{T(a)}{T(a-1)} = \frac{(a+1)^{a+1}(s-a+1)^{s-a+1}}{a^a(s-a+2)^{s-a+2}}$$

$$
\begin{aligned}
&= \left(\frac{a+1}{a}\right)^a \left(\frac{s-a+1}{s-a+2}\right)^{s-a+2} \frac{a+1}{s-a+1} \\
&= \left(1+\frac{1}{a}\right)^a \left(1-\frac{1}{s-a+2}\right)^{s-a+2} \frac{a+1}{s-a+1} \\
&\leq e^1 e^{-1} \frac{a+1}{s-a+1} \\
&\leq \frac{a+1}{s-a+1} \\
&\leq \frac{\frac{s}{2}+1}{\frac{s}{2}+1} \leq 1
\end{aligned}
$$

Therefore $T(a) \leq T(a-1)$. By induction, this means $T(a) \leq T(1)$ for all $1 \leq a \leq \lfloor \frac{s}{2} \rfloor$. We also have a symmetry: $T(a) = T(s-a)$, so this means $T(a) \leq T(1)$ for $\lceil \frac{s}{2} \rceil \leq a \leq s-1$ as well. Therefore:

$$
\begin{aligned}
\sum_{a=1}^{s-1} T(a) &\leq \sum_{a=1}^{s-1} T(1) = (s-1)2^2 s^s \\
&\leq 4s^{s+1} = 4\left(\frac{s}{s+1}\right)^{s+1} (s+1)^{s+1} = 4\left(1-\frac{1}{s+1}\right)^{s+1} (s+1)^{s+1} \\
&\leq \frac{4}{e}(s+1)^{s+1}
\end{aligned}
$$

□

Lemma 9. *Let $U(s,q)$ be the set of sequences of positive integers, where $T \in U$ if and only if $|T| = q$ and $\sum_{1 \leq i \leq q} T_i = s$, where $s \geq q \geq 1$. Then $\sum_{T \in U(s,q)} \prod_{1 \leq i \leq q} (T_i+1)^{T_i+1} \leq \left(\frac{4}{e}\right)^{q-1} (s+1)^{s+1}$*

Proof. We proceed by induction on the length of the sequences. For $q = 1$, U contains a single sequence T with $T_1 = s$. Then $\sum_{T \in U(s,q)} \prod_{1 \leq i \leq q} (T_i + 1)^{T_i+1} = (s+1)^{s+1} = \left(\frac{4}{e}\right)^0 (s+1)^{s+1}$.

Assume that the theorem holds for all sequences of length $q \geq 1$. We will show that it also holds for all sequences of length $q+1$.

$$
\sum_{T \in U(s,(q+1))} \prod_{1 \leq i \leq q+1} (T_i+1)^{T_i+1} \leq \sum_{T_1=1}^{s-q} (T_1+1)^{T_1+1} \sum_{T' \in U((s-T_1),q)} \prod_{1 \leq i \leq q} (T'_i+1)^{T'_i+1}
$$

Applying our inductive hypothesis gives:

$$\leq \sum_{a=1}^{s-1} (a+1)^{a+1} \left(\frac{4}{e}\right)^{q-1} (s-a+1)^{s-a+1}$$
$$\leq \left(\frac{4}{e}\right)^{q-1} \sum_{a=1}^{s-1} (a+1)^{a+1} (s-a+1)^{s-a+1}$$

Using Lemma 8

$$\leq \left(\frac{4}{e}\right)^{q} (s+2)^{s+2}$$

□

Lemma 10. *For $a > 0$,*

$$\sum_{x=1}^{\infty} \left(\frac{a}{x}\right)^x \leq 2ae^{\frac{a}{e}}$$

Proof. Over the positive reals the function $f(x) = \left(\frac{a}{x}\right)^x$ is maximized at $x = \frac{a}{e}$, for which it has value $e^{\frac{a}{e}}$. Therefore:

$$\begin{aligned}
\sum_{x=1}^{\infty} \left(\frac{a}{x}\right)^x &\leq \sum_{x=1}^{2a-1} \left(\frac{a}{x}\right)^x + \sum_{x=2a}^{\infty} \left(\frac{a}{x}\right)^x \\
&\leq \sum_{x=1}^{2a-1} (e^{\frac{a}{e}}) + \sum_{x=2a}^{\infty} \left(\frac{1}{2}\right)^x \\
&\leq e^{\frac{a}{e}}(2a-1) + 1 \\
&\leq 2ae^{\frac{a}{e}}
\end{aligned}$$

□

7 A Lower Bound

We now show the following lower bound on the number of elements n in terms of the security parameter N, such that cuckoo hashing with a stash can fail with negligible probability in N. For consistency with other parts of the paper, we use the 2-table construction but this can easily be adapted to other constructions.

Theorem 13. *If $n = \mathcal{O}(\log(N))$ and $n - s = \Omega(n)$ then it is impossible for a 2-table Cuckoo Hash table to have a negligible build failure probability in N.*

Proof. Since $n - s = \Omega(n)$, it follows that $n - s \geq c_0 n$ for sufficiently large n where the constant c_0 satisfies $0 < c_0 \leq 1$. Therefore:

$$\begin{aligned}
\frac{n-s}{n} &\geq c_0 \\
\frac{n-s-2}{n} &\geq c_0 - \frac{2}{n} \\
\frac{n-s-2}{n} &\geq \frac{c_0}{2} \text{ when } n \geq \frac{4}{c_0} \\
\frac{n-s-2}{n} &\geq c_1 \text{ for constant } c_1 \text{ satisfying } 0 < c_1 \leq \frac{1}{2}
\end{aligned}$$

Since $n = \mathcal{O}(\log(N))$, there is some constant c_2 such that $n \leq c_2 \log(N)$ (for sufficiently large n).

Let $m = dn$ be the size of each table.

If all n items are hashed to the first $\lceil \frac{n-s-2}{2} \rceil$ locations in both tables, then $2\lceil \frac{n-s-2}{2} \rceil \leq n - s - 1$ items can be stored in the table, and s items can be stored in the stash, but 1 item will not be able to be stored at all, so the build fails.

The probability that all n items are stored in the first $\lceil \frac{n-s-2}{2} \rceil$ locations in both tables is at least:

$$\begin{aligned}
\left(\frac{n-s-2}{2dn}\right)^{2n} &\geq \left(\frac{c_1}{2d}\right)^{2c_2 \log(N)} \\
&\geq N^{2c_2 \log\left(\frac{c_1}{2d}\right)}
\end{aligned}$$

This is non-negligible in N. Therefore the probability of a build failure is non-negligible.

□

This immediately implies the contrapositive:

Corollary 3. *Cuckoo Hashing with a stash requires $n - s = o(n)$ or $n = \omega(\log(N))$ in order to succeed with failure negligible in N.*

The case that $n - s = o(n)$ is very unnatural–it implies that a sub-constant number of elements are stored in the table, at which point the Cuckoo table is not providing much use. Thus, in any realistic setting where Cuckoo tables are used, it is necessary that $n = \omega(\log(N))$. This provides the lower bound for n in terms of N such that Cuckoo Hashing with a stash has a negligible probability of failure.

8 Acknowledgments

I would like to thank Sampath Kannan for helpful discussions.

This research was sponsored in part by ONR grant (N00014-15-1-2750) "SynCrypt: Automated Synthesis of Cryptographic Constructions" and a gift from Ripple Labs, Inc.

References

[ADW14] Martin Aumüller, Martin Dietzfelbinger, and Philipp Woelfel. Explicit and efficient hash families suffice for cuckoo hashing with a stash. *Algorithmica*, 70(3):428–456, 2014.

[AKL+20] Gilad Asharov, Ilan Komargodski, Wei-Kai Lin, Kartik Nayak, Enoch Peserico, and Elaine Shi. OptORAMa: Optimal oblivious RAM. In *EUROCRYPT*, pages 403–432. Springer, 2020.

[Aum10] Martin Aumüller. *An alternative analysis of cuckoo hashing with a stash and realistic hash functions*. PhD thesis, Diplomarbeit, Technische Universität Ilmenau, 2010.

[DK12] Michael Drmota and Reinhard Kutzelnigg. A precise analysis of cuckoo hashing. *ACM Transactions on Algorithms (TALG)*, 8(2):1–36, 2012.

[Dwa69] Meyer Dwass. The total progeny in a branching process and a related random walk. *Journal of Applied Probability*, 6(3):682–686, 1969.

[GM11] Michael T Goodrich and Michael Mitzenmacher. Privacy-preserving access of outsourced data via oblivious RAM simulation. In *ICALP*, pages 576–587. Springer, 2011.

[GMOT12] Michael T Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. Privacy-preserving group data access via stateless oblivious RAM simulation. In *SODA*, pages 157–167. SIAM, 2012.

[KLO12] Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. On the (in) security of hash-based oblivious RAM and a new balancing scheme. In *SODA*, pages 143–156. SIAM, 2012.

[KM10] Achim Klenke and Lutz Mattner. Stochastic ordering of classical discrete distributions. *Advances in Applied probability*, 42(2):392–410, 2010.

[KM19] Eyal Kushilevitz and Tamer Mour. Sub-logarithmic distributed oblivious RAM with small block size. In *PKC*, pages 3–33. Springer, 2019.

[KMW09] Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More robust hashing: Cuckoo hashing with a stash. *SIAM Journal on Computing*, 39(4):1543–1561, 2009.

[Kut10] Reinhard Kutzelnigg. A further analysis of cuckoo hashing with a stash and random graphs of excess r. *Discrete Mathematics and Theoretical Computer Science*, 12(3):81–101, 2010.

[LO13] Steve Lu and Rafail Ostrovsky. Distributed oblivious RAM for secure two-party computation. In *TCC*, pages 377–396. Springer, 2013.

[MP23] Brice Minaud and Charalampos Papamanthou. Generalized cuckoo hashing with a stash, revisited. *Information Processing Letters*, page 106356, 2023.

[Pit98] Jim Pitman. Enumerations of trees and forests related to branching processes and random walks. *Microsurveys in discrete probability*, 41:163–180, 1998.

[PR01] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In *ESA*, pages 121–133. Springer, 2001.

[PR10] Benny Pinkas and Tzachy Reinman. Oblivious RAM revisited. In *CRYPTO*, pages 502–519. Springer, 2010.

[PSSZ15] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 515–530, 2015.

[PSWW18a] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based psi via cuckoo hashing. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 125–157. Springer, 2018.

[PSWW18b] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based psi via cuckoo hashing. *Cryptology ePrint Archive*, 2018.

[Wie16] Udi Wieder. Hashing, load balancing and multiple choice draft, 2016.

[Yeo22] Kevin Yeo. Cuckoo hashing in cryptography: Optimal parameters, robustness and applications. *Cryptology ePrint Archive*, 2022.