3.5-Way Cuckoo Hashing for the Price of 2-and-a-Bit

Eric Lehman[1] and Rina Panigrahy[2]

[1] Google, Mountain View, CA. Email: elehman@google.com
[2] Microsoft Research, Mountain View, CA. Email: rina@microsoft.com

Abstract. The study of hashing is closely related to the analysis of balls and bins; items are hashed to memory locations much as balls are thrown into bins. In particular, Azar et. al. [2] considered putting each ball in the less-full of two random bins. This lowers the probability that a bin exceeds a certain load from exponentially small to doubly exponential, giving maximum load $\log \log n + O(1)$ with high probability. Cuckoo hashing [20] draws on this idea. Each item is hashed to two buckets of capacity k. If both are full, then the insertion procedure moves previously-inserted items to their alternate buckets to make space for the new item. In a natural implementation, the buckets are represented by partitioning a fixed array of memory into non-overlapping blocks of size k. An item is hashed to two such blocks and may be stored at any location within either one. We analyze a simple twist in which each item is hashed to two *arbitrary* size-k memory blocks. (So consecutive blocks are no longer disjoint, but rather overlap by $k-1$ locations.) This twist increases the space utilization from $1-(2/e+o(1))^k$ to $1-(1/e+o(1))^{1.59k}$ in general. For $k=2$, the new method improves utilization from 89.7% to 96.5%, yet lookups access only two items at each of two random locations. This result is surprising because the opposite happens in the non-cuckoo setting; if items are not moved during later insertions, then shifting from non-overlapping to overlapping blocks makes the distribution less uniform.

1 Introduction

The study of hashing is closely related to the analysis of balls and bins; items are hashed to memory locations much as balls are thrown into bins. Simple twists on balls and bins processes have produced surprising observations and led to breakthroughs in hashing methods.

In particular, it is well-known that if n balls are thrown into n bins independently and randomly, then the largest bin gets $(1+o(1)) \ln n / \ln \ln n$ balls with high probability. Azar et. al. [2] showed that assigning each ball to the less-full of two random bins makes the final distribution far more uniform. In fact, the probability that a bin exceeds a certain load drops from exponentially small to doubly exponential. This leads to the concept of two-way hashing, where the most-loaded bucket gets $\log \log n + O(1)$ items with high probability. So dramatic is this improvement that it can be used in practice to efficiently implement hash lookups in packet routing hardware [5]. The lookup operation must search for an item in two buckets, but these operations can be parallelized in hardware by placing two different hash tables in separate memory components. More generally, if each item is hashed to $d \geq 2$ buckets, then the maximum load improves to $\log \log n / \log d + O(1)$.

Cuckoo hashing [20, 12] extends two-way hashing by moving previously-inserted items to their alternate buckets to make space for a new item. Pagh and Rodler [20] showed that even with buckets of capacity one, moving items during inserts gives a space utilization of 50% with high probability. Several generalizations of cuckoo hashing perform even better. Fotakis et al [12] suggested hashing each item to $d > 2$ buckets, and Dietzfelbinger and Weidling [9] suggested using buckets with capacity $k > 1$. One appealing choice is to hash items to $d = 2$ buckets of capacity $k = 2$, which gives 89.7% [21, 14, 6] space utilization. (The latter two references improve upon the earlier, weaker estimate.) More generally, the analysis of cuckoo hashing is related to the appearance of dense subgraphs in random graphs. For example, the space utilization achieved by cuckoo hashing where items are hashed to $d = 2$ buckets of capacity k is directly related to the threshold at which a dense subgraph appears in the random graph $G(n, p)$. The space utilization is precisely p/k where p is the threshold at which a dense subgraph with ratio of edges to vertices exceeding k appears. Analysis of this threshold in [14, 6] implies a space utilization of about $1 - (2/e + o(1))^k$.

In a natural implementation of two-way cuckoo hashing, the buckets are represented by partitioning a fixed array of memory into non-overlapping blocks of size k. Each item is hashed to two such blocks and may be stored at any of the $2k$ memory locations within those blocks. This implementation avoids expensive dynamic memory allocation. Furthermore, a lookup searches just two contiguous memory segments, which is highly desirable in practice. For example, after an initial read from a random location in main memory or on a disk, subsequent bytes can often be read orders of magnitude faster. (This is a heuristic, not a certainty; for example, the extra bytes might lie beyond a cached portion of memory.) And all $2k$ memory locations can be probed in parallel in a hardware implementation.

We suggest another simple twist that significantly improves space utilization while preserving the desirable property of only two random memory accesses per lookup. Previously, the hash table memory was partitioned into disjoint blocks of size k. Now, we regard every set of k consecutive memory locations as a block. So consecutive blocks are no longer disjoint, but rather overlap by $k - 1$ memory locations. As before, each item is hashed to two blocks and may be stored at any memory location within those blocks. We show that this simple change improves the space utilization from $1 - (2/e + o(1))^k$ to $1 - (1/e + o(1))^{1.59k}$. Experimentally, we demonstrate that space utilization improves from 89.7% to 96.5% in the practically-important case where each item is hashed to two blocks of capacity $k = 2$. This result is surprising because the opposite happens in the non-cuckoo setting; if items are not moved during later insertions, then shifting from non-overlapping to overlapping blocks actually makes the distribution of items less uniform.

2 Related Work

Balls and bins analysis still continues to produce surprising results. Vöcking [24] observed that asymmetry helps in load balancing. If each ball is mapped to d bins with equal load, then the ball should be inserted in the leftmost bin. With this simple change,

the maximum load drops to $O(\log \log n/d)$. He also showed that breaking ties in this way is the best possible policy to minimize the maximum load.

Berenbrink *et al.* [4] extended the balls and bins analysis to the case where the number of balls m is greater than the number of bins n, showing that the difference in the height of the minimum and maximum bin is independent of m. Precisely, it is $\frac{\log \log n}{\ln d} + O(1)$. Corresponding results hold when ties are broken asymmetrically. Balls and bins on graphs has been analyzed in [17]. Weighted analysis of balls and bins was studied in [23]. Multi-choice hashing with a limited number of moves was studied in [21]. Extensive work has been done in the area of parallel balls and bins [1] and the related study of algorithms to emulate shared memory machines (as for example, PRAMs) on distributed memory machines (DMMs) [8, 7, 13, 22].

More recent works have studied the idea of using a small CAM (content-addressable memory) in conjunction with cuckoo hashing to lower the insert/delete time [18, 19]. Arbitman *et al.* [25] proved that it is possible to get constant time for all operations with about 50% space utilization by using a small auxiliary hash table. The appearance of dense subgraphs in random graphs was studied in [14, 6]. These build upon earlier works that investigate the appearance of a k-core – a subgraph with minimum degree at least k – in random graphs [3, 16].

Other related work includes the first static dictionary data structure with constant look up time by Fredman, Komlos and Szemeredi [15] that was generalized to a dynamic data structure by Dietzfelbinger et al. in [11] and [10]. In practice, however, these algorithms are more complex to implement than cuckoo hashing.

3 Our Contribution

We propose a new twist on multi-choice hashing that significantly improves memory utilization, yet accesses only two small regions of memory. Our main theorem compares this new variation to the algorithm analyzed in [14, 6]. In both algorithms, each item is hashed to two memory blocks of size k. The item may be stored at any location in either block, and previously-inserted items may be moved to their alternate locations to make space for the new item. The lookup operation searches the k locations in each of the two blocks associated with the item sought. The distinction is that the earlier ALG-DISJOINT-CUCKOO algorithm hashes items to only a restricted set of memory blocks; specifically, the hash table memory is initially partitioned into disjoint blocks of size k, and items are hashed only to those blocks. In our new twist, ALG-OVERLAP-CUCKOO, an item may be hashed to any two size-k memory blocks. Our main result states that this new algorithm has better space utilization for large k. Let α_k denote the utilization for ALG-CUCKOO-DISJOINT and β_k for ALG-CUCKOO-OVERLAP.

Theorem 1. *For large k, $\alpha_k < \beta_k$. Specifically,*

- $\alpha_k \leq 1 - (2/e - o(1))^k$
- $\beta_k \geq 1 - (1/e + o(1))^{(2-\gamma)k}$*, where γ is the maximum value of the function $-x + x\log(2(1 + 1/x))$, which is about* 0.41.

Experimentally, we show that memory utilization improves significantly for small, practical values of k as well. For example, $\alpha_2 = 89.7\%$ while $\beta_2 = 96.5\%$.

This result is surprising, because the opposite effect is observed in the "non-cuckoo" setting; that is, when previously-inserted items are not allowed to be moved to make space for a new item. Again, we compare two algorithms. In both cases, there are n balls and n bins in a line. For each ball, we randomly pick two blocks of k consecutive bins and throw the ball into the least-loaded bin in the less-loaded block. As before, ALG-NOMOVE-DISJOINT uses only blocks from an initial, disjoint partition, while ALG-NOMOVE-OVERLAP uses all blocks. In this case, using overlapping blocks actually leads to a less uniform distribution:

Theorem 2. *[17] For large k,*

- *with high probability, ALG-NOMOVE-DISJOINT gives a maximum load on a bin of $O(\log \log n/k)$ and*
- *with high probability, ALG-NOMOVE-OVERLAP gives a maximum load on a bin of $\Omega(\log \log n/ \log k)$.*

Intuition: Here is a simple intuition as to why overlapping blocks give higher space utilization for cuckoo-hashing than disjoint blocks. Consider the case $k = 2$ with disjoint blocks. Note that in cuckoo-hashing, we perform a breadth-first-search for an empty bin by first looking at the two blocks where a new ball hashes. If these are full, then we look at the 4 alternate blocks where the 4 balls in those blocks could go. Continuing recursively, we visit 2^k blocks at a depth of k. Thus the search tree is binary for ALG-CUCKOO-DISJOINT. We will argue that this search tree has a slightly higher degree for ALG-CUCKOO-OVERLAP, which uses overlapping blocks. The faster this search tree branches, the more likely we are to find an empty bin before getting stuck; that is, before reaching leaves whose potential children are all already in the tree. In the ALG-CUCKOO-OVERLAP variant, the two balls in a full block B can potentially be moved to other bins besides those in their alternate blocks. This happens if one of the blocks of these balls overlaps partially with B – in this case, such a ball can also be displaced to the other bin in this partially-overlapping block. Thus the branching factor of the search tree is slightly more than 2. We will demonstrate this phenomena in the experiment section.

Our analysis assumes that each item is hashed to two blocks in a single hash table. But essentially the same analysis applies to the case where an item is hashed to one block in each of two, separate tables.

4 Theoretical Analysis

We will now prove the main theorem 1. We are comparing two cuckoo-based algorithms that access two random blocks of size k each. Algorithm ALG-CUCKOO-DISJOINT accesses from a collection of disjoint blocks at offsets that are multiples of k; whereas ALG-CUCKOO-OVERLAP picks blocks at random offsets. Let us say we have nk bins and we are adding balls one by one till we overflow.

For any balls and bins process, a configuration of balls and bins can be viewed as a hypergraph G where each bin is a node and each ball is a hyperedge connecting its bin choices. The following lemma is well known.

Lemma 3. *For any cuckoo algorithm (ALG-CUCKOO-DISJOINT or ALG-CUCKOO-OVERLAP) a set of ball insertions succeeds iff there is no subgraph with more hyperedges then vertices.*

Remark 4. For ALG-CUCKOO-DISJOINT, we can think of a block as a single vertex. Thus α_k corresponds to the number of edges in a random graph $G(n, p)$ when a subgraph with density (ratio of edges to vertices) more than k appears.

Proof. The "only if" part is straightforward. For the "if" part, consider the case when a ball insertion fails. We will look at another bipartite graph with balls on one side and bins on another and an edge between them if the ball is allowed to be placed in the bin. So each ball has degree k. Further, we mark an edge between a ball and a bin red if the ball actually chooses that bin and blue otherwise. Now, for a new ball insert, if there is an alternating path of red and blue edges that leads to an empty bin, then we can successfully insert the ball using a sequence of cuckoo moves. So a ball insert fails iff all bins reachable through such alternating paths are full. In such a case, look at the set of bins reachable using such alternating paths from a new ball whose insertion failed. This set of bins is a subgraph with more balls than bins because all bins have a ball plus there is the new ball that could not be inserted. This subgraph of the bipartite graph corresponds to a subgraph in the hypergraph G with more hyperedges than nodes. □

Our main theorem follows from the following two claims. We will sometimes drop the subscript k for convenience. Let $\overline{\alpha}, \overline{\beta}$ denote $1 - \alpha$ and $1 - \beta$ respectively. Similarly for other variables.

Claim. [6, 16] $\alpha_k \leq 1 - (2/e - o(1))^k$

Proof. Although the exact threshold for α_k has been computed before in [6, 16], we present a simpler analysis of the asymptotic formula. For a random graph on n nodes with $k(1 - \overline{\alpha})n$ edges, let us find the fraction f of nodes that have degree at most $k - 1$. This degree distribution of the nodes is given by the Poisson distribution with mean $2k(1 - \overline{\alpha})$ and f is at least the fraction of nodes with degree exactly $k - 1$.

$$
\begin{aligned}
f &\geq e^{-2k(1-\overline{\alpha})} \frac{(2k(1 - \overline{\alpha}))^{k-1}}{(k-1)!} \\
&= e^{2k\overline{\alpha}} \frac{1}{2(1 - \overline{\alpha})} (1 - \overline{\alpha})^k e^{-2k} \frac{(2k)^k}{k!} \\
&= e^{k\overline{\alpha}} \frac{1}{k^{O(1)}} e^{-2k} \frac{(2k)^k}{(k/e)^k} \\
&= e^{k\overline{\alpha}} k^{-O(1)} (2/e)^k
\end{aligned}
$$

So if we ignore this f fraction of the nodes, the remaining $(1 - f)n$ nodes have at least $k(1 - \overline{\alpha})n + fn$ edges; let us check when the density (ratio of edges to vertices) in

the remaining subgraph is more than k. This happens if $k(1-\overline{\alpha})n + fn > k(1-f)n$ or $(k+1)f > k\overline{\alpha}$ or $f > \frac{k}{k+1}\overline{\alpha}$

Plugging in the previous expression for f, we need $e^{k\overline{\alpha}}k^{-O(1)}(2/e)^k > \frac{k}{k+1}\overline{\alpha}$ or $\overline{\alpha} < k^{-O(1)}e^{k\overline{\alpha}}(2/e)^k = e^{k\overline{\alpha}}(2/e+o(1))^k$

Clearly $\overline{\alpha} = (2/e - o(1))^k$ satisfies this. Note that although we used the expected value of f, the actual fraction is concentrated close to this with high probability by Chernoff bounds. □

Next we will show a lower bound for β_k. We will make use of the following simple claims.

Claim. If $0 \le p \le q \le 1$ and q is allowed to vary, the function $q\log(\frac{p}{q}) + (1-q)\log(\frac{1-p}{1-q})$ is decreasing in q.

Proof. Taking derivative with respect to q, we get $\log(\frac{p}{q}) - \log(\frac{1-p}{1-q})$ which is negative. □

Claim. Let γ denote the maximum value of the function $g(x) = -x + x\log(2(1+1/x))$ when $x > 0$. Then $\gamma \le 0.41$.

The claim can be verified by a simple plot of the function.

Claim. $\beta_k \ge 1 - (1/e + o(1))^{(2-\gamma)k}$

Proof. We need to demonstrate a value of $\overline{\beta} = (1/e + o(1))^{(2-\gamma)k}$ for which there is almost surely no subgraph in the hypergraph with more edges than vertices. We will upper bound the probability of finding a subset of bins with more edges than bins and argue that this is unlikely with high probability. Consider nk nodes (bins) and βnk hyperedges (balls) where each hyperedge chooses two sets of k contiguous bins. Although in our algorithm the bins are arranged in a line, for simplicity of analysis we will think of them as arranged in a circle. Any subset of bins can be viewed as a union of contiguous regions in this circle. Look at a subset S of nodes, say it consists of $r = \epsilon n$ contiguous regions and suppose $t = \delta nk$ bins fall outside S. A sequence of k bins can be viewed as a segment of length k. If this is to lie in S, then both its endpoints must be in one of the r contiguous regions in S. Of the nk possible segments of length k at most $nk - r(k-1) - t = nk(1-\delta-\epsilon(1-1/k))$ lie in S. The probability that a hyperedge falls in the nodes in S is $p = (1-\delta-\epsilon(1-1/k))^2$. For S to have more edges than nodes, the required fraction of edges to fall in S is at least $q = (1-\delta)/\beta$ (note that $\delta \ge \overline{\beta}$). Let $x = \delta/\epsilon$.

First, a simple calculation will show that unless $\delta = O(1/\sqrt{k})$ and $\epsilon = \tilde{\Omega}(k\beta^2)$, the probability of finding such a high-density subgraph on S is exponentially small. To see that, note that the subset of $(1-\delta)nk$ bins in S is expected to get no more than $(1-\delta)^2\beta nk$ edges. To get $(1-\delta)nk$ edges, it has to get at least factor $\frac{1}{\beta(1-\delta)} \ge 1 + \delta + \overline{\beta}$ of the expectation. By Chernoff bounds, the probability this happens it at most $e^{-\Omega(\delta+\overline{\beta})^2 nk}$. The number of ways of choosing S is at most the number of ways of choosing the $2\epsilon n$ endpoints of the regions, which is $\binom{n}{2\epsilon n} \le (\frac{e}{2\epsilon})^{2\epsilon n} \le e^{2\epsilon\log\frac{e}{2\epsilon}n}$.

So the total probability is at most $e^{n(2\epsilon\log\frac{e}{2\epsilon}-k\Omega(\delta+\overline{\beta})^2)}$. This is exponentially small unless the exponent is nonnegative, which happens when $\epsilon\log(1/\epsilon)\geq k\Omega(\delta+\overline{\beta})^2$. Since $\delta>0$, we get $\epsilon>\tilde{\Omega}(k\beta^2)$. Also since $\epsilon\log(1/\epsilon)<1$, we get $k\delta^2\leq O(1)$ or $\delta\leq O(1/\sqrt{k})$.

Next, we will do a more detailed calculation of the probability. The number of ways of choosing the subset S is at most the number of ways of first choosing the r starting points of the r regions which is at most $\binom{nk}{\epsilon n}$, and then choosing the r closing points of the regions so that the total size of the r segments is δnk. This can be done in at most $\binom{\delta nk}{\epsilon n}$ ways.

For S to have more edges than nodes, it must get at least qN edges, where $N=n\beta k$ is the total number of edges and $q\geq(1-\delta)/\beta\geq 1-\delta$. The probability L, that a set S gets qN edges is $\binom{N}{qN}p^{qN}(1-p)^{N(1-q)}$. Taking natural log, we get.:

$$
\begin{aligned}
\log L &= \log\binom{N}{qN}+qN\log p+\overline{q}N\log\overline{p} \\
&\leq N(-q\log q-\overline{q}\log\overline{q})+qN\log p+\overline{q}N\log\overline{p} \\
&= N(q\log(p/q)+\overline{q}\log(\overline{p}/\overline{q})) \\
&= \beta nk(q\log(p/q)+\overline{q}\log(\overline{p}/\overline{q})) \\
&= \beta nka
\end{aligned}
$$

where $a=q\log(p/q)+\overline{q}\log(\overline{p}/\overline{q})$. Since, $1-\delta\leq q\leq 1$, a as a function of q is maximized when $q=1-\delta$, so

$$
\begin{aligned}
a &\leq (1-\delta)\log\frac{(1-\delta-\epsilon(1-1/k))^2}{1-\delta}+\delta\log\frac{2(\delta+\epsilon)}{\delta} \\
&\leq (1-\delta)(-2(\delta+\epsilon))-(1-\delta)\log(1-\delta)+\delta\log\frac{2(\delta+\epsilon)}{\delta} \\
&\leq -2(1-\delta)(\delta+\epsilon)+\delta+\delta\log\frac{2(\delta+\epsilon)}{\delta} \\
&\leq (1-\delta)(-2\epsilon-2\delta)+(1-\delta)(\delta+\delta\log\frac{2(\delta+\epsilon)}{\delta}) \\
&\leq (1-o(1))(-2\epsilon-\delta+\delta\log\frac{2(\delta+\epsilon)}{\delta}) \\
&\leq (1-o(1))\epsilon(-2-x+x\log 2(1+1/x))
\end{aligned}
$$

The total log probability of finding some component S that has more edges than vertices is at most

$$
\begin{aligned}
&\log[\binom{nk}{n\epsilon}\binom{\delta nk}{\epsilon n}L] \\
&\leq \log\binom{nk}{n\epsilon}+\log\binom{\delta nk}{\epsilon n}+nk\beta a \\
&= n\epsilon\log\frac{ek}{\epsilon}+n\epsilon\log\frac{e\delta k}{\epsilon}+nk(1-o(1))\epsilon(-2-x+x\log 2(1+1/x)) \\
&= n\epsilon\log\frac{ek}{\epsilon}+n\epsilon\log ekx+nk(1-o(1))\epsilon(-2-x+x\log 2(1+1/x))
\end{aligned}
$$

$$= nf$$

where

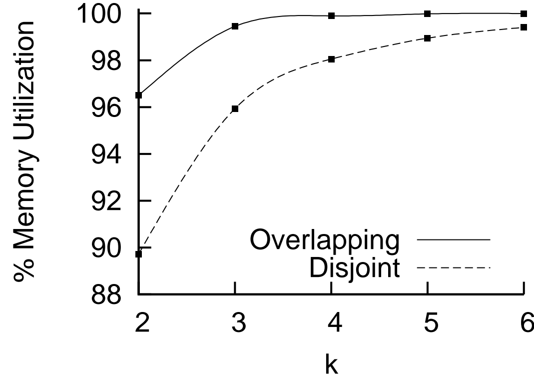
$$\begin{aligned}
f &= \epsilon(\log \frac{ek}{\epsilon} + \log(ekx) + k(1-o(1))(-2-x+x\log 2(1+1/x))) \\
&= \epsilon(\log \frac{e^2k^2}{\epsilon} + k(1-o(1))(o(x)-2-x+x\log 2(1+1/x))) \\
&\le \epsilon(\log \frac{e^2k^2}{\epsilon} + k(1-o(1))(-2+\gamma+o(1))) \\
&= \epsilon(\log \frac{e^2k^2}{\epsilon} + k(1-o(1))(-2+\gamma)).
\end{aligned}$$

Observe that the expression bounding f is independent of n. So if for a given β for all ϵ and δ, f is negative and less than some fixed value independent of n, then it means that the probability that there is a high density set S is exponentially small. Also ϵ cannot be arbitrarily small as we know that it is $\tilde{\Omega}(k\beta^2)$. Now for f to be non negative we need $\log \frac{e^2k^2}{\epsilon} > (1-o(1))k(2-\gamma)$ or $\epsilon < e^{-(1-o(1))(2-\gamma)k}$. We also need to add the probability over all possible values of r and t for S but there are only n^2 possible values which cannot compensate for an exponentially decreasing function; so w.h.p. no such set S exists.

Also from the expression $f = \epsilon(\log \frac{e^2k^2}{\epsilon} + k(1-o(1))(o(x)-2-x+x\log 2(1+1/x))) \le \epsilon(\log \frac{e^2k^2}{\epsilon} + k(1-o(1))(-x+x\log 2+x\log(1+1/x)) \le \epsilon(\log \frac{e^2k^2}{\epsilon} + k(1-o(1))(-x\log e/2+1)$ it is clear that for f to be non negative $x \le O(\frac{1}{k}\log \frac{e^2k^2}{2\epsilon})$ giving $\delta \le O(\frac{1}{k}\log \frac{e^2k^2}{2\epsilon})\epsilon \le e^{-(1-o(1))(2-\gamma)k}$. This gives, $\overline{\beta} \le \delta < e^{-(1-o(1))(2-\gamma)k} \le (1/e+o(1))^{(2-\gamma)k}$. □

5 Experiments

Experiments suggest that using overlapping blocks improves memory utilization substantially even for small k. This implies that our twist gives a significant practical improvement. The situation is summarized in the figure below:



For the basic case of buckets with capacity $k = 2$, memory utilization increases from 89.7% to 96.5% when overlapping is allowed. For larger k, the overlapping scheme rapidly approaches full memory utilization: 99.44% for $k = 3$ and 99.90% for $k = 4$. Each percentage is from twenty trials using hash tables with an absolute capacity of 2^{20} items and "random" hash functions based on a cryptographic-quality pseudorandom number generator. Items were inserted into the hash table one-by-one until some item could not be added. The results were notably stable. In each case, the standard deviation was a few hundredths of a percent, so error bars would be invisible in the diagram. Such strongly-predictable behavior is appealing from a practical standpoint.

Other experimental data gives additional insight into this performance gap. Recall that the cuckoo insertion algorithm performs a breadth-first search for an empty location in the hash table. The table below shows the number of search tree nodes at each depth for a typical insertion using various hashing schemes. The structure of these trees is random in all cases, but much more noticeably for the new scheme, so three examples are given for that case:

	# of Nodes at Depth					
Each Item is Hashed to:	1	2	3	4	5	6
3 buckets, capacity 1	3	6	12	24	48	96
4 buckets, capacity 1	4	12	36	108	324	972
2 disjoint buckets, capacity 2	4	8	16	32	64	128
2 overlapping blocks, capacity 2	4	11	24	55	136	330
same as above	4	9	20	49	122	305
same as above	4	10	24	62	150	364

For example, if each item is hashed to 4 buckets of capacity 1, as on the second line, then the root of the search tree has 4 children. Items occupying the corresponding locations can each move to 3 other slots, so nodes lower in the search tree have 3 children each. This gives the sequence 4, 12, 36, 108, 324, 972. In contrast, suppose each item is hashed to two disjoint buckets of capacity $k = 2$ as on the third line. Again, the root of the search tree has 4 children. But items in those slots can be moved to at most *two* slots not already explored. Thus, lower nodes in the search tree have only 2 children instead of 3.

The last three lines show data for the new scheme introduced here. Once again, the root has 4 children. But now items occupying those location can be moved to either *two or three* other slots with equal probability. Thus, the width of the search tree is greater than for the disjoint-buckets approach and lies somewhere between the 3- and 4-bucket schemes shown on the first two lines. Since the new scheme accesses only two small, contiguous regions of memory, one might regard it as 3.5-way cuckoo hashing for the price of 2-and-a-bit.

References

1. M. Adler, S. Chakrabarti, M. Mitzenmacher, and L. Rasmussen. Parallel randomized load balancing. *In Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, pages 238–247, May 1995.

2. Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced allocations. *SIAM Journal on Computing, 29:180-200, 1999.A preliminary version of this paper appeared in Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing, 1994.*
3. S. Spencer B. Pittel and N. Wormald. Sudden emergence of a giant k-core in a random graph. *J. Combin. Theory Ser. B 67 (1996), no. 1*, pages 111–151.
4. Petra Berenbrink, Artur Czumaj, Angelika Steger, and Berthold Vöcking. Balanced allocations: The heavily loaded case. *SIAM J. Comput.*, 35(6):1350–1385, 2006.
5. A. Broder and M. Mitzenmacher. Using multiple hash functions to improve ip lookups. *Proceedings of IEEE INFOCOM 2001*, pages 1454–1463, 2001.
6. Julie Anne Cain, Peter Sanders, and Nick Wormald. The random graph threshold for k-orientiability and a fast algorithm for optimal multiple-choice allocation. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 469–476, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.
7. A. Czumaj, F. Meyer auf der Heide, and V. Stemann. Shared memory simulations with triple-logarithmic delay. *Lecture Notes in Computer Science, 979*, pages 46–59, 1995.
8. M. Dietzfelbinger and F. Meyer auf der Heide. Simple efficient shared memory simulations. *Proc. of the 5th SPAA (1993)*, pages 110–119.
9. M. Dietzfelbinger and P. Woelfel. Almost random graphs with simple hash functions. *35th STOC*, pages 629–638, 2003.
10. Martin Dietzfelbinger and Friedhelm Meyer auf der Heide. A new universal class of hash functions and dynamic hashing in real time. *In Proceedings of the 17th International Colloquium on Automata, Languages and Program- ming (ICALP '90), volume 443 of Lecture Notes in Computer Science*, pages 6–19. Springer–Verlag, Berlin, 1990.
11. Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Heide, Hans Rohnert, and Robert E. Tarjan. Dynamic perfect hashing: Upper and lower bounds. *SIAM J. Comput., 23(4)*, pages 738–761, 1994.
12. Peter Sanders Dimitris Fotakis, Rasmus Pagh and Paul Spirakis. Space efficient hash tables with worst case constant access time. *20th Annual Symposium on Theoretical Aspects of Computer Science, 2003.*
13. C. Scheideler F. Meyer auf der Heide and V. Stemann. Exploiting storage redundancy to speed up randomized shared memory simulations. *Theoretical Computer Science, 162(2):245-281, 1996. Preliminary version in Proc. of the 12th STACS (1995)*, pages 267–278.
14. Daniel Fernholz and Vijaya Ramachandran. The k-orientability thresholds for gn, p. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 459–468, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.
15. Michael L. Fredman, Janos Komlos, and Endre Szemeredi. Storing a sparse table with o(1) worst case access time. *J. Assoc. Comput. Mach., 31(3)*, pages 538–544, 1984.
16. Svante Janson and Malwina J. Luczak. A simple solution to the k-core problem. *Random Struct. Algorithms*, 30(1-2):50–62, 2007.
17. Krishnaram Kenthapadi and Rina Panigrahy. Balanced allocation on graphs. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 434–443, New York, NY, USA, 2006. ACM.
18. A. Kirsch and M. Mitzenmacher. Using a queue to de-amortize cuckoo hashing in hardware. *45th Allerton Conference on Communication, Control, and Compu-ting*, pages 751–758, 2007.
19. A. Kirsch, M. Mitzenmacher, and U. Wieder. More robust hashing: Cuckoo hashing with a stash. *16th ESA, pages 611-622, 2008.*

20. R. Pagh and F. Rodler. Cuckoo hashing. *Journal of Algorithms 51 (2004), p. 122-144. A preliminary version appeared in proceedings of the 9th Annual European Symposium on Algorithms.*, pages 121–133, 2001.
21. Rina Panigrahy. Efficient hashing with lookups in two memory accesses. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 830–839, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.
22. Jan H. M. Korst Peter Sanders, Sebastian Egner. Fast concurrent access to parallel disks. *Algorithmica, 35(1). A Preliminary version appeared in SODA 2000*, pages 21–55,2003.
23. Kunal Talwar and Udi Wieder. Balanced allocations: the weighted case. In *STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 256–265, New York, NY, USA, 2007. ACM.
24. Berthold Vöcking. How asymmetry helps load balancing. *J. ACM*, 50(4):568–589, 2003.
25. Moni Naor Yuriy Arbitman and Gil Segev. De-amortized cuckoo hashing: Provable worst-case performance and experimental results. 2009.