Pseudo-Random Number Generation for Sketch-Based Estimations

FLORIN RUSU and ALIN DOBRA
University of Florida

The exact computation of aggregate queries, like the size of join of two relations, usually requires large amounts of memory – constrained in data-streaming – or communication – constrained in distributed computation – and large processing times. In this situation, approximation techniques with provable guarantees, like sketches, are one possible solution. The performance of sketches depends crucially on the ability to generate particular pseudo-random numbers. In this paper we investigate both theoretically and empirically the problem of generating k-wise independent pseudo-random numbers and, in particular, that of generating 3 and 4-wise independent pseudo-random numbers that are fast range-summable (i.e., they can be summed-up in sub-linear time). Our specific contributions are: (a) we provide a thorough comparison of the various pseudo-random number generating schemes, (b) we study both theoretically and empirically the fast range-summation property of the 3 and 4-wise independent generating schemes, (c) we provide algorithms for the fast range-summation of two 3-wise independent schemes, BCH and Extended Hamming, (d) we show convincing theoretical and empirical evidence that the Extended Hamming scheme performs as well as any 4-wise independent scheme for estimating the size of join of two relations using AMS-sketches, even though it is only 3-wise independent. We use this scheme to generate estimators that significantly outperform the state-of-the-art solutions for two problems – *size of spatial joins* and *selectivity estimation*.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems—*Query processing*; G.3 [**Probability and Statistics**]: Random number generation

General Terms: Algorithms, Experimentation, Performance, Theory

Additional Key Words and Phrases: Sketches, data synopses, approximate query processing, fast range-summation

1. INTRODUCTION

The exact computation of aggregate queries usually requires large amounts of memory – constrained in data-streaming – or communication – constrained in distributed computation – and large processing times. In this situation, approximation techniques with provable guarantees that can be maintained over data-streams or that can be used for estimations in distributed environments are the only viable solution.

This is an extended version of the paper *Fast range-summable random variables for efficient aggregate estimation* that was published in the Proceedings of ACM SIGMOD 2006 Conference.
Material in this paper is based upon work supported by the National Science Foundation under grant number NSF-CAREER-IIS-0448264.
Authors' email addresses: Florin Rusu (frusu@cise.ufl.edu); Alin Dobra (adobra@cise.ufl.edu).

Due to their linearity, AMS-sketches [Alon et al. 1996] have all these properties and they have been successfully used for the estimation of aggregates, like the size of join, over data-streams [Alon et al. 2002; Dobra et al. 2002] and in distributed environments, like sensor networks [Kempe et al. 2003]. AMS-sketches depend on the ability to efficiently generate large families of random variables with particular properties: limited degree of independence and, for applications in which the input is specified as a set of intervals, *fast range-summation* (i.e., the ability to sketch an interval in time sub-linear in its size). As we show in this paper, the last property is especially useful when AMS-sketches are applied for solving problems like *size of spatial joins* [Das et al. 2004], and *selectivity estimation* for dynamic construction of histograms [Thaper et al. 2002].

In this paper we investigate both theoretically and empirically the known methods for generating the random variables used by AMS-sketches with the goal of identifying the generating schemes that are practical, both for the traditional application of AMS-sketches, i.e., aggregate computation over streaming data, and for applications that involve interval input. More specifically, our contributions are:

—We provide a thorough comparison of the various generating schemes with the goal of identifying the efficient ones. To this end, we explain how the schemes can be implemented on modern processors and we use such implementations to empirically evaluate them.

—We provide a detailed study of the practicality of fast range-summation for the known generating schemes. We show that no 4-wise independent generating scheme is practically fast range-summable, even though the scheme based on Reed-Muller codes can be theoretically range-summed in sub-linear time [Calderbank et al. 2005].

—We provide a formal treatment of dyadic intervals that allows us to design efficient fast range-summable algorithms for two schemes: 3-wise independent BCH scheme (BCH3) [Alon et al. 1986] and Extended Hamming scheme (EH3) [Feigenbaum et al. 2002].

—We provide thorough theoretical analysis for the behavior of the BCH3 and EH3 schemes as replacements for the 4-wise independent schemes in size of join estimations using AMS-sketches. We show that while BCH3 is a poor replacement for non-skewed distributions, EH3 is in general as good as or significantly better than any 4-wise independent scheme. The fact that EH3 gets within a constant factor of the error of the 4-wise independent schemes for the problem of computing the L^1-difference of two streaming vectors was theoretically proved in [Feigenbaum et al. 2002]. Here we generalize this result to size of join estimations using AMS-sketches and we show that EH3 can always replace the 4-wise independent schemes without sacrificing accuracy. Our empirical evaluation confirms these theoretical predictions.

—We explain how two problems, size of spatial joins and selectivity estimation, can be reduced to the size of join problem, reduction that allows us to provide estimates of their results based on AMS-sketches. The resulting estimators significantly outperform the state-of-the-art solution based on dyadic mappings [Das et al. 2004], sometimes by as much as a factor of 8 in relative error.

In the rest of the paper, we first give some introductory notions in Section 2 with an emphasis on dyadic intervals. Then we discuss the known generating schemes for random variables with limited independence in Section 3. In Section 4 we investigate which generating schemes are fast range-summable from a practical point of view and we design fast range-summable algorithms for the BCH3 and EH3 schemes. In Section 5 we provide theoretical proof that the Extended Hamming (EH3) generating scheme works as well as the 4-wise independent schemes with the added benefit that it is fast range-summable. We provide empirical evidence for this fact and a thorough comparison between EH3 and the 4-wise schemes on two applications with previously proposed solutions in Section 6. We conclude in Section 7.

2. PRELIMINARIES

In this section we give some preliminaries that are useful for understanding the rest of the paper. We begin by revisiting the original AMS-sketching technique for computing the size of join of two point relations. Then we identify a class of applications that can be reduced to size of join problems involving an interval relation and a point relation. In order to apply the AMS-sketching technique for the derived problem, there exist two alternatives: either use random variables that are fast range-summable, or apply a domain transformation for reducing the size of the intervals. Since the second alternative is the state-of-the-art solution for sketching intervals, we examine it in this section. Fast range-summable random variables are analyzed in detail in Section 4.

2.1 AMS-Sketches

AMS-sketches are randomized schemes that were initially introduced for approximating the second frequency moment of a relation in small space [Alon et al. 1996]. Afterwards, they were applied to the general size of join problem [Alon et al. 2002].

The size of join of two relations R and S, each with a single attribute A, consists in computing the quantity $|R \bowtie_A S| = \sum_{i \in I} r_i s_i$, where I is the domain of A and r_i and s_i, respectively, are the frequencies of the tuples in the two relations. The exact solution to this problem is to maintain the frequency vectors $\bar{r}$ and $\bar{s}$ and then to compute the size of join. Such a solution would not work if the amount of memory or the communication bandwidth is smaller than $Min(|I|, \textit{cardinality of relation})$.

The approximate solution based on sketches is defined as follows [Alon et al. 2002]:

(1) Start with a family of 4-wise independent ± 1 random variables ξ corresponding to the elements in the domain I.
(2) Define the sketches $X_R = \sum_{i \in I} r_i \xi_i = \sum_{t \in R} \xi_{t.A}$ and $X_S = \sum_{i \in I} s_i \xi_i = \sum_{t \in S} \xi_{t.A}$. The sketch summarizes the frequency vector of the relation as a single value by randomly projecting each tuple over the values $+1$ and -1. Notice that the tuples with the same value are always projected either on $+1$ or -1.
(3) Let the random variable X be $X = X_R X_S$. X has the properties that it is an unbiased estimator for the size of join $|R \bowtie_A S|$ and that it has small variance. An estimator with relative error at most ϵ with probability at least $1 - \delta$ can

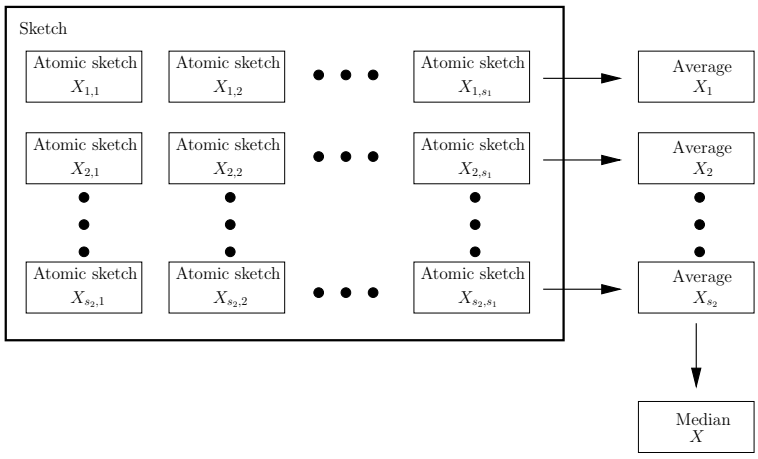

Fig. 1. AMS-Sketches. Atomic sketches $X_{i,j}$ correspond to the random variable X. Average X_i corresponds to the average of s_1 atomic sketches $X_{i,j}$, $1 \leq j \leq s_1$. Median X corresponds to the median of s_2 Average X_i, $1 \leq i \leq s_2$.

be obtained by taking the median over averages of multiple independent copies of the random variable X: the median is computed over $s_2 = 2\log\frac{1}{\delta}$ such averages, each average containing $s_1 = \frac{8}{\epsilon^2}\frac{\mathrm{Var}(X)}{E[X]^2}$ instances of X (Figure 1).

Sketches are perfectly suited for both data-streaming and distributed computation since they can be updated on pieces. For example, if the tuples in one relation are streamed one by one, the atomic sketch can be computed by simply adding the value ξ_i corresponding to the value i of the current item. For distributed computation, each party can compute the sketch of the data it owns. Then, by only exchanging the sketch with the other parties and adding them up, the sketch of the entire dataset can be computed.

The AMS-sketches described above have at their core ± 1 random variables. The values $+1$ and -1 can be obtained by simply generating random bits and then interpreting them as -1 and $+1$ values. The necessary and sufficient requirement for X to be an unbiased estimator for the size of join of R and S is that the ξ family of random variables to be 2-wise independent. The stronger 4-wise independence property is required in order to make the variance as small as possible, thus reducing the number of copies of X that need to be averaged in order to achieve the desired accuracy. But, as we will see in Section 3, generating 4-wise independent random variables is more demanding both in the amount of required memory, as well as in time efficiency.

2.2 Sketch Applications

The original AMS-sketching technique was subsequently extended and applied to other problems. [Dobra et al. 2002] show how to extend AMS-sketches to compute complex aggregates over general equi-joins. [Ganguly et al. 2004] show how to improve the AMS-sketches error guarantees by using *skimmed sketches*. [Charikar et al. 2004] introduce *count sketches* as a sketch variant with improved update

time. AMS-sketches have been also used as building blocks in many applications. For example, [Ganguly et al. 2003] compute aggregates over expressions involving set operators, while [Feigenbaum et al. 2002] approximate the L^1-difference of two data-streams. A solution based on AMS-sketches for the wavelet decomposition of a data-stream is introduced in [Gilbert et al. 2003], while dynamic quantiles are approximated using *random subset sums* in [Gilbert et al. 2005]. In networking, sketches can be applied for change detection as in [Krishnamurthy et al. 2003].

Although they were introduced for estimating the size of join of two point relations, AMS-sketches are a versatile approximation method that can accommodate multiple types of input. In this paper we focus on a variation of the original size of join problem in which one of the relations is specified as intervals (this is equivalent to specifying every point inside the interval). To motivate the importance of the derived problem, we introduce two applications that can be expressed as the size of join of two relations, one consisting of points, and one containing intervals.

2.2.1 *Size of Spatial Joins.* Given two relations of line segments in the unidimensional space, the size of spatial join problem is to compute the number of segments from the two relations that overlap. The approach in [Das et al. 2004], even though not explicitly stated, reduces the size of spatial join problem to two size of join problems, each involving a point relation and an interval relation: the size of join of the line segments from the first relation and the segment end-points from the second relation and, symmetrically, the size of join of the segment end-points from the first relation and the segments from the second relation.

In order to implement a solution based on AMS-sketches for the point-interval size of join problem, we have to deal first with efficiently sketching intervals. The solution proposed in [Das et al. 2004] realizes a number of transformations for reducing the size of each interval, thus improving the sketching time. We present a detailed analysis of this generic method for sketching intervals in Section 2.4.

2.2.2 *Selectivity Estimation for Building Dynamic Histograms.* Any histogram construction algorithm has to evaluate the average frequency of the elements in a bucket, also known as the selectivity of the bucket. By introducing a virtual relation having value 1 only for the elements in the bucket, and value 0 otherwise, the average frequency can be expressed as the size of join between the original relation and the virtual relation corresponding to the bucket, divided by the size of the bucket. Usually a bucket is an interval in the unidimensional space, thus the problem of sketching intervals has to be solved in order to implement an AMS-sketches solution for the selectivity estimation problem.

[Thaper et al. 2002] introduce a solution based on sketches for building dynamic histograms. The streaming relation is summarized as a sketch and then the histogram is extracted from the sketch. The authors focused more on how to determine the buckets such that the resulting histogram to be optimal. They did not consider in detail the problem of sketching a bucket and took this step as a black-box in their algorithms.

2.3 Problem Formulation

The common point of the above applications is that they can be reduced to size of join problems between a point relation and an interval relation. In order to

implement effective sketch-based solutions for this derived problem, the sketch of an interval has to be computed more efficiently than sketching each point in the interval.

Summarizing, the problem we treat in this paper is to **estimate the size of join between a point relation and an interval relation using AMS-sketches**. We start by revisiting the state-of-the-art solution based on dyadic mappings (DMAP). Before introducing solutions that use fast range-summable random variables, we first take a close look at the known schemes for generating ± 1 random variables that are at the core of the AMS-sketching technique.

2.4 Dyadic Mapping (DMAP)

The dyadic mapping method [Das et al. 2004], which we call here DMAP, uses dyadic intervals in order to reduce the size of an interval, thus making possible the efficient sketching of intervals. Since dyadic intervals are at the foundation of the method, we introduce them first. We will see that dyadic intervals are also important for fast range-summation (Section 4).

2.4.1 *Dyadic Intervals.* [Gilbert et al. 2005] introduce dyadic ranges for sketching intervals in the context of quantile computation. Although they are employed in different work [Gilbert et al. 2003; Feigenbaum et al. 2002; Das et al. 2004], a formal treatment of dyadic intervals was not previously provided. Such a formal treatment is necessary to design efficient algorithms for methods that need dyadic interval decomposition. In this section we provide such a formal treatment.

For the rest of this section, consider that the domain I has size $|I| = N = 2^n$. If this is not the case, the domain can be extended to the first power of 2 that is greater than N. The dyadic intervals of the domain I are intervals that can be organized in a hierarchical structure – the dyadic intervals over the domain $\{0, \ldots, 15\}$ are depicted in Figure 2 – that has $n + 1$ layers. There exists only one interval at level 0 – the entire domain. On the subsequent levels, each interval is split in two equal-size intervals to produce smaller dyadic intervals. On the last level, all dyadic intervals have size one and they consist of all the individual points in the domain. Intuitively, such a construction produces intervals that have size a power of 2 (more precisely all the intervals at level k have size 2^{n-k}) and that have boundaries aligned at multiples of powers of 2. This description is good enough to argue properties of dyadic intervals, but it does not suffice to formally prove the correctness of algorithms that use dyadic intervals. Insights from the strict theoretical treatment of dyadic intervals actually lead to very efficient implementations.

We start our presentation with a formal definition of dyadic intervals and some basic properties.

Definition 2.1. A **dyadic interval** over the domain $I = \{0, 1, \ldots, N-1\}$, $|I| = N = 2^n$, is an interval of the form $[q2^j, (q+1)2^j)$, where $0 \leq j \leq n$ and $0 \leq q \leq 2^{n-j} - 1$.

PROPOSITION 2.2. *There are exactly 2^j dyadic intervals at level j, each containing 2^{n-j} points from I.*

PROOF. Follows directly from definition. □

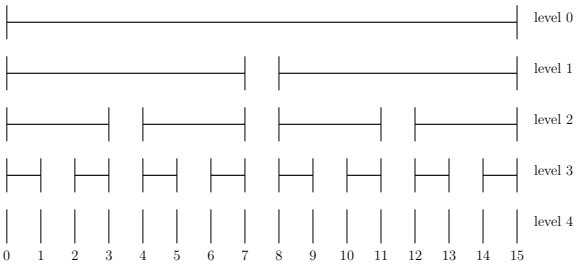

Fig. 2. The set of dyadic intervals over the domain $I = \{0, 1, \ldots, 15\}$.

PROPOSITION 2.3. *The dyadic intervals at level j, $0 \leq j \leq n$, form a partition of the domain I. That is, they are disjoint and their union is equal with the entire domain.*

PROOF. If j is fixed, using the above definition, the dyadic intervals at level j are: $[0, 2^j), [2^j, 2 \cdot 2^j), \ldots, [(2^{n-j} - 1) \cdot 2^j, 2^{n-j} \cdot 2^j)$. Clearly, they form a partition of $[0, 2^n)$. □

PROPOSITION 2.4. *Let δ_1 and δ_2 be two arbitrary distinct dyadic intervals. If $\delta_1 \cap \delta_2 \neq \emptyset$, then either $\delta_1 \subset \delta_2$ or $\delta_2 \subset \delta_1$.*

PROOF. Let $\delta_1 = [q2^j, (q+1)2^j)$ and $\delta_2 = [r2^k, (r+1)2^k)$. We have two distinct cases, $j = k$, and $j \neq k$. We discuss each in turn.

Suppose $j = k$. Then either $q = r$, which is not possible since $\delta_1 \neq \delta_2$, or $q \neq r$. In the latter case, the two dyadic intervals, according to Proposition 2.3, are different elements of a partition of the domain, so they do not intersect, which is a contradiction.

Suppose, without loss of generality, that $j > k$. We have to show that $\delta_2 \subset \delta_1$. This is the only possibility since δ_1 has more elements than δ_2. We have two distinct cases: $r2^k \in \delta_1$ and $r2^k < q2^j$.

—If $r2^k \in \delta_1$ then either $r2^k \leq (q+1)2^j - 2^k$, in which case $\delta_2 \subset \delta_1$ since we can fit all 2^k elements of δ_2 from $r2^k$ to the end of δ_1, or $r2^k \in ((q+1)2^j - 2^k, (q+1)2^j)$. In this last case, we have $r2^k > (q+1)2^j - 2^k$ and $r2^k < (q+1)2^j$. Dividing the two inequalities by 2^k, which is a positive quantity, we get: $(q+1)2^{j-k} > r > (q+1)2^{j-k} - 1$. Since $j > k$, 2^{j-k} is an integer quantity. But both r and q are integers and this inequality is a contradiction since we cannot insert an integer between two consecutive integers.

—If $r2^k < q2^j$, since $\delta_1 \cap \delta_2 \neq \emptyset$, it must be the case that $(r+1)2^k > q2^j$. From these two inequalities, by dividing by 2^k, we have: $r < q2^{j-k} < r + 1$. This is again a contradiction since $q2^{j-k}$ is an integer that has to be between two consecutive integers.

□

An interesting question is how to express an arbitrary interval as the union of dyadic intervals. Since the decomposition is not unique in general and it is desirable to have decompositions of small sizes, we look for minimal decompositions in terms

of the number of elements. The concept of minimal dyadic cover of an interval and its properties are introduced next.

Definition 2.5. The **minimal dyadic cover** of an interval $[\alpha, \beta]$, $D([\alpha, \beta])$, is the set of dyadic intervals $\delta_1, \delta_2, \ldots, \delta_m$ of smallest cardinality, for which $\delta_1 \cup \delta_2 \cup \cdots \cup \delta_m = [\alpha, \beta]$.

PROPOSITION 2.6. *The dyadic intervals in a minimal dyadic cover of an interval form a partition of that interval.*

PROOF. Let $\delta_1 \cup \delta_2 \cup \cdots \cup \delta_m = [\alpha, \beta]$ be a minimal dyadic cover of the arbitrary interval $[\alpha, \beta]$. It is enough to show that no two dyadic intervals intersect. According to Proposition 2.4, if two dyadic intervals intersect, then one includes the other. The one that is included can be eliminated from the minimal dyadic cover of $[\alpha, \beta]$ without changing the coverage, which implies that the cover is not minimal, a contradiction. □

Definition 2.7. $d([\alpha, \beta]) = q2^j$ is the **dyadic cut-point** of the interval $[\alpha, \beta]$ if j is the largest integer smaller than n ($|I| = 2^n$) for which there exists q such that $q2^j \in [\alpha, \beta]$.

PROPOSITION 2.8. *The dyadic cut-point of an interval is unique.*

PROOF. Let $q2^j$ and $q'2^j$, $q < q'$, be two distinct dyadic cut-points of the same interval. Since, by definition, both are included in the interval, all integers $q''2^j$ with $q \leq q'' \leq q'$ are also included in the interval, so they are also dyadic cut-points. Since $q \neq q'$, at least one of the q'' has to be even, for example $q'' = 2r$, thus $q''2^j = r2^{j+1}$. This contradicts the fact that j is maximal and proves the uniqueness of $q2^j$. □

Using these properties, we formalize the relation between the dyadic cut-point of an interval and its minimal dyadic cover in the following lemma.

LEMMA 2.9. *Let $[\alpha, \beta]$ be an interval, $d([\alpha, \beta]) = q2^j$ be its dyadic cut-point, and $D([\alpha, \beta])$ be its minimal dyadic cover. Then:*

(1) *None of the dyadic intervals in $D([\alpha, \beta])$ contains the dyadic cut-point, except as a left end-point.*
(2) *The minimal dyadic cover $D([\alpha, \beta])$ is the union of the minimal dyadic covers of $[\alpha, q2^j)$ and $[q2^j, \beta]$, i.e., $D([\alpha, \beta]) = D([\alpha, q2^j) \cup D([q2^j, \beta])$.*
(3) *The minimal dyadic cover $D([q2^j, \beta])$ consists of a sequence of at most j dyadic intervals with strictly decreasing sizes.*
(4) *The minimal dyadic cover $D([\alpha, q2^j)$ consists of a sequence of at most j dyadic intervals with strictly increasing sizes.*
(5) *The minimal dyadic cover $D([\alpha, \beta])$ contains at most $2j$ dyadic intervals, from which at most two are from the same level.*

PROOF.

(1) Let δ be a dyadic interval in the minimal dyadic cover that contains the dyadic cut-point $d([\alpha, \beta]) = q2^j$, but not as its left end-point. Then, by Proposition 2.4, $[q2^j, (q+1)2^j) \subset \delta$. Equality is not possible because $q2^j$ would be at

the start of δ. Then $[\frac{q}{2}2^{j+1}, (\frac{q}{2}+1)2^{j+1}) \subset \delta$ is the smallest dyadic interval that can contain the previous interval. This means that $\frac{q}{2}2^{j+1}$ is included in $[\alpha, \beta]$, which contradicts the fact that $q2^j$ is the dyadic cut-point.

(2) Suppose the union of the minimal dyadic covers of $[\alpha, q2^j)$ and $[q2^j, \beta]$ is not the minimal dyadic cover of $[\alpha, \beta]$ and let $D'([\alpha, \beta])$ be this minimal cover. If $q2^j$ is not included in the core of any of the dyadic intervals in the minimal dyadic cover, then we can partition the minimal dyadic cover into two sets, one that covers $[\alpha, q2^j)$, and one that covers $[q2^j, \beta]$. These covers have to be minimal, otherwise the cover of $[\alpha, \beta]$ is not minimal. But, according to (1), $q2^j$ cannot be part of the core of any dyadic interval in any dyadic cover of $[\alpha, \beta]$, so it has to be the case that the statement is correct.

(3) We give a constructive proof of the statement. The claim is that the following algorithm finds the minimal dyadic cover of $[q2^j, \beta]$. Starting from the point $q2^j$, find the largest dyadic interval that fits into $[q2^j, \beta]$, and then iterate from the point immediately following this dyadic interval until all of $[q2^j, \beta]$ is covered. Since $[q2^j, \beta]$ is finite and each dyadic interval contains at least one point, the algorithm terminates in at most $\beta - q2^j + 1$ steps.
Let us show that the built cover is minimal and has size at most j. We consider an arbitrary interval $[q'2^{j'}, \beta]$ with $q'2^{j'}$ being its dyadic cut-point. Let $q''2^{j''}$ be the dyadic cut-point of the interval $(q'2^{j'}, \beta]$, i.e., the next dyadic cut-point since the current dyadic cut-point is excluded from the interval. We claim that the interval $[q'2^{j'}, q''2^{j''})$ is the largest dyadic interval that is part of the minimal dyadic cover of $[q'2^{j'}, \beta]$. First, $j'' < j'$ otherwise $q''2^{j''}$ would be the dyadic cut-point of $[q'2^{j'}, \beta]$. This means that $[q'2^{j'}, q''2^{j''})$ is a dyadic interval of size $2^{j''}$ since it is equivalent to $[2^{j'-j''}q' \cdot 2^{j''}, q''2^{j''})$ and it cannot be a union of dyadic intervals from level j'' because this would contradict the fact that $q''2^{j''}$ is a dyadic cut-point. In fact, this dyadic interval is the largest dyadic interval included in $[q'2^{j'}, \beta]$, otherwise we again contradict the fact that $q''2^{j''}$ is a dyadic cut-point. From this, it follows directly that the dyadic interval $[q'2^{j'}, q''2^{j''})$ is the unique largest dyadic interval in the minimal dyadic cover of $[q'2^{j'}, \beta]$. By induction, the sequence of dyadic intervals generated by the algorithm produces the minimal dyadic cover of $[q2^j, \beta]$. Since $j'' < j'$, the algorithm ends in at most j steps.

(4) The proof is symmetric to (3).

(5) Follows directly from (3) and (4).

□

COROLLARY 2.10. *The minimal dyadic cover of an interval* $[\alpha, \beta]$ *has cardinality at most* $2(n-1)$*, where* $|I| = 2^n$*. That is,* $|D([\alpha, \beta)]| \leq 2(n-1)$.

PROOF. Follows directly from Lemma 2.9 by observing that $j = n-1$ maximizes the cardinality of the minimal dyadic cover. □

The algorithm used in the constructive proof of Lemma 2.9 uses as a prerequisite the dyadic cut-point and starts determining the minimal dyadic cover to the left and to the right of this point. To avoid determining the dyadic cut-point, we can run the algorithm in reverse and find the dyadic intervals in the minimal dyadic

cover starting from α and going to the right, and starting from β and going to the left. Moreover, it is enough to inspect the binary representation of α and β to determine these dyadic intervals, as it is shown in Algorithm *MinimalDyadicCover*. For example, to determine the dyadic intervals covering $[\alpha, d([\alpha, \beta]))$, we inspect the bits of α starting from the least significant and whenever we detect a 1 bit, say in position j, we have to include the dyadic interval of size 2^j that covers all the integers that have the same binary representation as α on the most significant $n - j$ bits. For β the same pattern has to be followed, but the 0 bits signal the inclusion of a dyadic interval. At every step, a new bit in the description of α and β is analyzed and the algorithm stops when $\alpha = \beta + 1 = d([\alpha, \beta])$. Example 2.11 illustrates how the algorithm runs.

MinimalDyadicCover($[\alpha, \beta]$)

```
1  j ← 0
2  D([α,β]) ← ∅
3  while α ≤ β
4        do if α_j = 1
5              then
6                   D([α,β]) ← D([α,β]) ∪ [α, α+2^j)
7                   α ← α+2^j
8           if β_j = 0
9              then
10                  D([α,β]) ← D([α,β]) ∪ [β−2^j+1, β+1)
11                  β ← β−2^j
12          j ← j+1
13 return D([α,β])
```

Example 2.11. *Consider the interval* $[100, 200]$*, with* $\alpha = 100 = (01100100)_2$ *and* $\beta = 200 = (11001000)_2$*. The algorithm inspects the bits in* α *and* β *starting from the least significant one. If a* 1 *bit is seen in* α*, useful processing has to be done. The same is equivalent for* 0 *bits in* β*. Bit* 0 *in* β *is* 0*, implying the addition to the minimal dyadic cover of the interval* $[200, 201)$ *and the new* β *becomes* $\beta^{(1)} = 199 = (11000111)_2$*. The first bit in* α *that is equal with* 1 *is the third one. It forces the addition of the interval* $[100, 104)$ *to the minimal dyadic cover and* $\alpha^{(1)} = 104 = (01101000)_2$*. The next steps of the algorithm add the intervals* $[104, 112), [192, 200), [112, 128)$ *and* $[128, 192)$ *to the minimal dyadic cover,*

$$D([100, 200]) = \{[100, 104), [104, 112), [112, 128), [128, 192), [192, 200), [200, 201)\}$$

At this point, $\alpha^{(3)} = 128 = (10000000)_2$ *and* $\beta^{(3)} = 127 = (01111111)_2$*, and the algorithm returns. Notice that* $\alpha^{(3)}$ *equals the dyadic cut-point of the interval,* $d([100, 200]) = 2^7 = 128$*.*

Algorithm *MinimalDyadicCover* has running time $O(\log(\beta - \alpha))$ if the elementary operations on integers take $O(1)$ time (as it is the case for implementations on real computers). If the notion of dyadic cut-point is not developed and Lemma 2.9 is not proved, the intuitive algorithm to find the minimal dyadic cover of an interval

is: find the largest dyadic interval contained in $[\alpha, \beta]$; add this dyadic interval to the minimal dyadic cover and then recurse on the start and end left-over parts of $[\alpha, \beta]$. The largest dyadic interval contained in a given interval can be found in $O(\log(|I|)$ steps and this has to be done for each of $O(\log(\beta - \alpha))$ dyadic intervals in the decomposition of $[\alpha, \beta]$, resulting in an $O(\log(|I|)\log(\beta - \alpha))$ algorithm. Notice that there is an $O(\log(|I|)$ gap between the two algorithms. This is due to the fact that the algorithm we provide cleverly avoids the search for the maximal dyadic interval and it is able to determine what dyadic intervals are in the minimal cover by simply inspecting the bit representation of the end-points α and β.

The main conclusion of the above mathematical formalization is that any interval can be decomposed efficiently in at most a logarithmic number of dyadic intervals. This implies that instead of identifying an interval by all the points it contains, we can represent the interval by its minimal dyadic cover. Thus, a reduction from a linear-size representation to a logarithmic-size representation is obtained. As we will see, this is a significant improvement in the sketching context. Moreover, determining the minimal dyadic cover of an interval can be implemented very efficiently by considering only the binary representation of the interval end-points.

2.4.2 *DMAP Method.* DMAP [Das et al. 2004] employs dyadic intervals for efficiently sketching intervals in the context of the size of join between a point relation and an interval relation. As already mentioned, the dyadic representation of an interval is more compact than the point representation.

DMAP uses this fact and is based on a set of three transformations (Figure 3). The original domain is mapped into the domain of all possible dyadic intervals that can be defined over it. An interval in the original domain is mapped into its minimal dyadic cover in the new domain. By doing this, the size of the interval reduces to at most a logarithmic number of points in the new domain, i.e., the number of sketch updates reduces from linear to logarithmic in the size of the interval. At the same time, a point in the original domain maps to the set of all dyadic intervals that contain the point in the new domain, thus increasing the number of sketch updates from one to logarithmic in the size of the original domain.

[Das et al. 2004] prove that the size of join between a point relation and an interval relation is invariant to these transformations, that is, the size of join over the original domain is equal with the size of join over the mapped domain. The intuition behind the proof lies on the fact that for each point in an interval there exists exactly one dyadic interval in the minimal dyadic cover that contains that point. Thus, by applying the AMS-sketching technique in the mapped domain, the size of join of the original relations is correctly approximated with the added benefit that the sketch of each relation is efficiently computed since both for an interval, as well as a point, at most $\log |I| = n$ dyadic intervals have to be sketched.

3. GENERATING SCHEMES

Based on the published literature [Alon et al. 1996], the AMS-sketches described in Section 2 need ± 1 4-wise independent random variables that can be generated in small space in order to produce reliable estimations. We address the question of whether the 4-wise independence is actually required later in the paper, but the 2-wise independence is a minimum requirement, since otherwise the estimator is

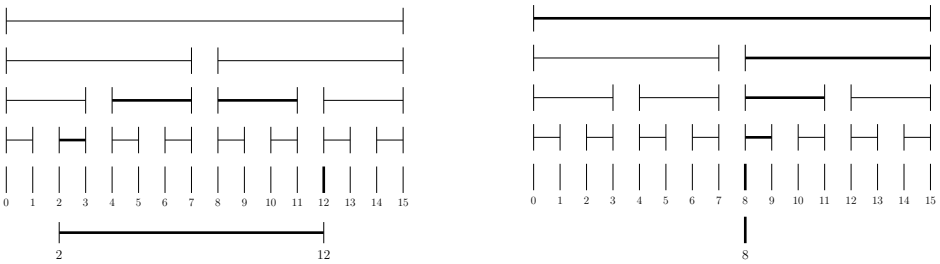

Fig. 3. Dyadic mappings. Domain $I = \{0, 1, \ldots, 15\}$ is mapped to the domain of all possible dyadic intervals that can be defined over I. Interval $[2, 12]$ is mapped to its minimal dyadic cover $D([2, 12]) = \{[2, 4), [4, 8), [8, 12), [12, 13)\}$ (left). Point 8 is mapped to the set of all dyadic intervals over I that contain it, i.e., $\{[8, 9), [8, 10), [8, 12), [8, 16), [0, 16)\}$ (right).

not even unbiased. In this section we review a number of generating schemes for random variables with limited degree of independence and we discuss how they can be implemented on modern processors. In the subsequent sections we refer back to these generating schemes.

3.1 Problem Definition

Let I be a domain – without loss of generality we assume that $I = \{0, 1, \ldots, N-1\}$ – and ξ be a family of two-valued random variables with a member for each $i \in I$, denoted by ξ_i. We assume that these random variables take the values ± 1 with the same probability. For different values, a mapping has to be defined. Usually, the domain I is large, for example $N = 2^{32}$, so a large amount of space is required to store even one instance of the family ξ. In order to be space efficient, each ξ_i is generated on demand according to a generating function:

$$\xi_i(S) = (-1)^{f(S,i)}, \quad i \in I \tag{1}$$

S is a random seed that can be stored in small space and the function f can be efficiently computed from the index i and S. The space used by S and the function f depend upon the desired degree of independence k. To allow simple generation, the seed S is uniformly and randomly chosen from its domain. With S generated this way, the k-wise independence property required for ξ translates into requiring that function f, for each k different indices i_1, i_2,...,i_k, generates any of the 2^k possible outcomes the same number of times, as S covers its domain. [Calderbank et al. 2005] give a formal definition of uniform k-wise independent random variables, also known as k-universal hashing functions:

Definition 3.1. A family ξ of ± 1 random variables defined over the sample space I is **k-wise uniform independent** if for any k different instances of the family $\xi_{i_1}, \xi_{i_2}, \ldots, \xi_{i_k}$, and any k ± 1 values $v_1, v_2, \ldots, v_k$, it holds:

$$Pr[\xi_{i_1} = v_1 \wedge \xi_{i_2} = v_2 \wedge \cdots \wedge \xi_{i_k} = v_k] = \frac{1}{2^k}$$

3.2 Orthogonal Arrays

The problem of generating two-valued k-wise independent random variables is equivalent to the construction of *orthogonal arrays*. Precisely, ± 1 k-wise indepen-

0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Table I. Orthogonal array $\mathrm{OA}(8, 4, 2, 3)$.

dent random variables over the domain $I = \{0, 1, \ldots, N-1\}$ form an orthogonal array with $|S|$ runs, N variables, two levels and strength k, denoted $\mathrm{OA}(|S|, N, 2, k)$, where $|S|$ is the size of the seed space. Since there exist clear relations defining the size of an orthogonal array, first we overview their main properties. Then, we extend these relations for characterizing the size of the seed space. Our presentation follows the approach in [Hedayat et al. 1999].

Definition 3.2. An $r \times c$ matrix A with entries from a set M is called an **orthogonal array** $\mathrm{OA}(r, c, m, t)$ with m levels, strength t and index λ – for some t in the range $0 \leq t \leq c$ – if every $r \times t$ sub-matrix of A contains each t-tuple based on M exactly λ times as a row.

M is the set containing the elements $\{0, 1, \ldots, m-1\}$, where m is the number of symbols or the number of levels. The number of rows r is known as the size of the array or the number of runs. c, the number of columns, represents the number of constraints, factors, or variables. A less formal way of defining an orthogonal array is to say that it is an array with the property that in any t columns you see equally often all possible t-tuples. When each tuple appears exactly once, i.e., $\lambda = 1$, we say that the array has index unity.

Example 3.3. *Table I represents an orthogonal array* $OA(8, 4, 2, 3)$ *that has index unity.*

Given the values of the parameters, the size of an orthogonal array is expressed in *Rao's Inequalities*. A particular form of these inequalities, for $m = 2$, was introduced in the context of k-wise independent random variables by [Alon et al. 1986]. We present the inequalities here and later we show how they are used for characterizing different construction schemes.

Theorem 3.4 [Hedayat et al. 1999]. *The parameters of an orthogonal array* $OA(r, c, s, t)$ *satisfy the following inequalities:*

$$r \geq \sum_{i=0}^{u} \binom{c}{i} (m-1)^i, \text{ if } t = 2u,$$

$$r \geq \sum_{i=0}^{u} \binom{c}{i} (m-1)^i + \binom{c-1}{u} (m-1)^{u+1}, \text{ if } t = 2u+1,$$

for $u \geq 0$, $r \equiv 0 \mod s^t$.

As mentioned, the problem of generating two-valued random variables that are k-wise independent is identical to constructing an orthogonal array $\mathrm{OA}(|S|, N, 2, k)$. In our particular case, we are interested in finding optimal constructions, that is, constructions that minimize the number of runs (rows) given fixed values for the number of variables (columns), i.e., $N = 2^n$, and the number of levels, since such constructions reduce the size of the seed, thus the space to generate and store it. Also, we need general constructions that exist for any value of the strength, i.e., the degree of independence.

The constructions based on BCH and Reed-Muller error-correcting codes are widely used. As stated in [Hedayat et al. 1999], the BCH arrays are the densest known for $N = 2^n$ variables, when n is odd. When n is even, there exist arrays with fewer runs based on Kerdock codes and, respectively, Delsarte-Goethals codes. However, these constructions exist only for even values of n and their advantage over the corresponding BCH arrays is minimal – one or two bits less for representing the number of runs – while overhead is incurred for computations in $\mathbb{Z}_4$, the ring of integers modulo 4. Both BCH and Reed-Muller constructions touch the bounds given by Rao's Inequalities only for small values of k, e.g., 2, 3.

3.3 Abstract Algebra

As mentioned previously, ± 1 random variables with limited independence can be obtained by generating $\{0, 1\}$ random variables and mapping them to ± 1. Since 0 and 1 are the only elements of the Galois Field with order 2, denoted by $GF(2)$, abstract algebra [Deskins 1996] is the ideal framework in which to talk about generating families of random variables with limited independence. The field $GF(2)$ has two operations: addition (boolean XOR) and multiplication (boolean AND). Abstract algebra provides two ways to extend the base field $GF(2)$ – vector spaces and extension fields – both of them useful for generating limited independence random variables.

$GF(2)^k$ *Vector Spaces* are spaces obtained by bundling together k dimensions, each with a $GF(2)$ domain. The only operation we are interested in here is the dot-product between two vectors $\bar{v}$ and $\bar{u}$, defined as: $\bar{v} \cdot \bar{u} = \oplus_{j=0}^{k-1} v_j \odot u_j$. For $GF(2)^k$ vector spaces this corresponds to AND-ing the arguments and then XOR-ing all the bits in the result.

$GF(p)$ *Prime Fields* are fields over the domain $\{0, 1, \ldots, p-1\}$ with both the multiplication and the addition defined as the arithmetic multiplication and addition modulo the prime p.

$GF(2^k)$ *Extension Fields* are fields defined over the domain $\{0, 1, \ldots, 2^k - 1\}$ that have two operations: addition, $+$, with zero element 0, and multiplication, $\cdot$, with unity element 1. Both addition and multiplication have to be associative and commutative. Also, multiplication is distributive over addition. All the elements, except 0, must have an inverse with respect to the multiplication operation. The usual representation of the extension fields $GF(2^k)$ is as polynomials of degree $k-1$ with the most significant bit as the coefficient for X^{k-1}, and the least significant as the constant term. The addition of two elements is simply the addition, term by term, of the corresponding polynomials. The multiplication is the polynomial multiplication modulo an irreducible polynomial of degree k that defines the extension field. With this representation, the addition is simple (just XOR the bit represen-

tations), but the multiplication is more intricate since it requires both polynomial multiplication and division.

As we will see, a large number of schemes use dot-products in vector spaces. Dot-products can be implemented by simply AND-ing the two vectors and XOR-ing the resulting bits. While AND-ing entire words (integers) on modern architectures is extremely fast, XOR-ing the bits of a word (which has to be eventually performed) is problematic since no high-level programming language supports such an operation (this operation is actually the parity-bit computation). To speed-up this operation, which is critical, we implemented it in *Assembly* for Pentium processors to take advantage of the supported 8-bit parity computation.

3.4 BCH Scheme

For all the schemes, we assume the domain to be $I = \{0, \ldots, 2^n - 1\}$, for a generic n (the description of the schemes depends on n). We also make the convention that $[a, b]$ is equivalent with the vector obtained by concatenating the vectors a and b. The size of a, b, and $[a, b]$ will be clear from the context.

The BCH scheme was first introduced in [Alon et al. 1986] and it is based on BCH error-correcting codes. This scheme can generate $(2k+1)$-wise independent random variables by using uniformly random seeds S that are $kn + 1$ bits in size. S can be represented as a vector $S = [s_0, S_0, \ldots, S_{k-1}]$, where s_0 is one bit and each S_i, $0 \leq i < k$, is a vector of size n. If s_0 is dropped from the seed, $2k$-wise independent random variables are obtained. The generating function $f(S, i)$ is defined as:

$$f(S, i) = S \cdot [1, i, \ldots, i^{2k-1}] \tag{2}$$

where i^{2k-1} is computed in the extension field $GF(2^n)$. This scheme comes close to the theoretical bounds [Hedayat et al. 1999] on how dense the seed space can be – it is the scheme with the densest seed requirement amongst all the known schemes. The proof that this scheme produces $(2k + 1)$-wise independent families can be found in [Alon et al. 1986].

Implementing i^{2k-1} over finite fields is problematic on modern processors if speed is paramount, but in the special case when only 3-wise independence is required this problem is avoided (see the speed comparison at the end of the section). Since the 3-wise independent version of this scheme, called BCH3 throughout the paper, is important in latter developments, we provide here its particular form:

$$f(S, i) = S \cdot [1, i] \tag{3}$$

The 4-wise independence required by the AMS-sketches can be obtained with $f(S, i) = S \cdot [1, i, i^3]$ (BCH5), which requires $(2n + 1)$-bit seeds, compared with $n + 1$ for BCH3.

Example 3.5. *We give an example that shows how the BCH schemes work. Consider that* $n = 16$ *and the seeds have the values:* $s_0 = 1$, $S_0 = 7,469 = (0001110100101101)_2$, *and* $S_1 = 346 = (0000000101011010)_2$. *We generate the* 3 *and* 5*-wise random variables corresponding to* $i = 2,500 = (0000100111000100)_2$. *Then,* $i^3 = 15,625,000,000 = (1001010001000000)_2$, *where the exponentiation is computed arithmetically, rather than over the extension field* $GF(2^{16})$, *and only the*

least significant 16 *bits are kept.*

$$S_0 \cdot i = \bigoplus_{j=0}^{15} \left(\begin{array}{c} 0001110100101101 \\ 0000100111000100 \end{array} \odot \right) = \bigoplus_{j=0}^{15} 0000100100000100 = 1$$

$$S_1 \cdot i^3 = \bigoplus_{j=0}^{15} \left(\begin{array}{c} 0000000101011010 \\ 1001010001000000 \end{array} \odot \right) = \bigoplus_{j=0}^{15} 0000000001000000 = 1$$

Computing $S_0 \cdot i$*, first the AND of the two binary values is calculated. Second, the sequential XOR of the resulting bits gives the final result. The same is equivalent for* $S_1 \cdot i^3$*. When implementing these schemes, the sequential XOR can be computed only at the end, after all the intermediate results are XOR-ed. The result for the* 3*-wise case is:*

$$s_0 \oplus S_0 \cdot i = 1 \oplus 1 = 0$$

while the result for the 5*-wise case is:*

$$s_0 \oplus S_0 \cdot i \oplus S_1 \cdot i^3 = 1 \oplus 1 \oplus 1 = 1$$

3.4.1 *Extended Hamming* 3*-Wise Scheme (EH3).* The Extended Hamming 3-wise scheme is a modification of BCH3 and it was introduced in [Feigenbaum et al. 2002]. It requires seeds S of size $n+1$ and its generating function is defined as:

$$f(S, i) = S \cdot [1, i] \oplus h(i) \tag{4}$$

where $h(i)$ is a nonlinear function of the bits of i. A possible form for h is:

$$h(i) = i_0 \vee i_1 \oplus \cdots \oplus i_{n-2} \vee i_{n-1}$$

Function h does not change the amount of independence, thus, from the traditional AMS-sketches theory, it is not as good as a 4-wise independent scheme. As we will show in Section 5, the use of EH3 actually results in size of join AMS-sketches estimations as precise as a 4-wise independent scheme gives[1].

From the point of view of a fast implementation, only a small modification has to be added to the implementation of BCH3 – the computation of function $h(i)$. As the experimental results show, there is virtually no running time difference between these schemes if a careful implementation is deployed on modern processors.

3.5 Reed-Muller Scheme

The BCH schemes require computations over extension fields for degrees of independence greater than 3. Since the AMS-sketches need 4-wise independent random variables, alternative schemes that require only simple computations might be desirable. The Reed-Muller scheme [Hedayat et al. 1999] generalizes the BCH codes in a different way in order to obtain higher degrees of independence. Seeds of size $1 + \binom{n}{1} + \cdots + \binom{n}{t}$ are required to obtain a degree of independence of $k = 2^{t+1} - 1$,

[1]The first theoretical result on the benefits of EH3 is provided in [Feigenbaum et al. 2002]. Our results are significantly more general.

$t > 0$. We introduce here only the 7-wise independent version of the scheme that requires $1 + n + \frac{n(n-1)}{2}$ seed bits:

$$f(S, i) = S \cdot [1, i, i^{(2)}] \tag{5}$$

where

$$i^{(2)} = [i_0 \odot i_1, i_0 \odot i_2, \ldots, i_{n-2} \odot i_{n-1}]$$

EXAMPLE 3.6. *We exemplify the Reed-Muller scheme (RM7), i.e., $t = 2$, for $n = 4$. $s_0 = 0$, $S_0 = 11 = (1011)_2$, and $S_1 = 45 = (101101)_2$. S_1 contains $\binom{4}{2} = 6$ bits. The index variable is $i = 6 = (0110)_2$. $i^{(2)} = [0 \odot 1, 0 \odot 1, 0 \odot 0, 1 \odot 1, 1 \odot 0, 1 \odot 0] = (000100)_2$.*

$$S_0 \cdot i = \bigoplus_{j=0}^{3} \begin{pmatrix} 1011 \\ 0110 \end{pmatrix}_{\odot} = \bigoplus_{j=0}^{3} 0010 = 1$$

$$S_1 \cdot i^{(2)} = \bigoplus_{j=0}^{5} \begin{pmatrix} 101101 \\ 000100 \end{pmatrix}_{\odot} = \bigoplus_{j=0}^{5} 000100 = 1$$

The final result is:

$$s_0 \oplus S_0 \cdot i \oplus S_1 \cdot i^{(2)} = 0 \oplus 1 \oplus 1 = 0$$

An important fact about the Reed-Muller scheme is that the seeds required are significantly large. For example, for a domain of size 2^{32}, RM7 needs seeds of $1 + 32 + \frac{32 \cdot 31}{2} = 529$ bits. For comparison, the BCH5 seeds are only $1 + 2 \cdot 32 = 65$ bits in size.

3.6 Polynomials over Primes Scheme

The generating schemes in the previous sections are derived from error-correcting codes. A different method to generate k-wise independent random variables uses polynomials over a prime number field.

The generating function is defined as in [Wegman and Carter 1981]:

$$f(S, i) = \sum_{j=0}^{k-1} a_j i^j \mod p \tag{6}$$

for some prime $p > i$ with each a_j picked randomly from $\mathbb{Z}_p$. The seed S consists of the k coefficients a_j, $0 \leq j < k$, each represented on $\lceil \log p \rceil$ bits. This scheme generates k-wise independent random variables that have p values. To obtain the ± 1 random variables, we can take the least significant bit in the binary representation and map it to ± 1[2].

The drawback of the polynomials scheme is that it requires multiplications over the field of the prime p, task that can be computationally intensive because it actually involves divisions and modulo operations. [Carter and Wegman 1979]

[2]This mapping introduces a small bias since the two values are not uniformly distributed. The bias is in the order of $\frac{1}{p}$, thus negligible.

introduce a method to increase the speed of these computations. The idea is to use Mersenne primes of the form $p = 2^l - 1$ which facilitate the computation of the remainder after a division with simple bitwise operations such as shifts, AND, OR, etc.

3.7 Toeplitz Matrices Scheme

The 2-universal Toeplitz family of 2^m-valued hash functions [Goldreich 1997; Bar-Yosseff et al. 2002] is defined over $m \times n$ Toeplitz matrices, i.e., matrices whose diagonals are homogeneous (all the entries in each diagonal contain the same value). Thus, a Toeplitz matrix is completely specified by the values in its first row and its first column. Each function $f : \{0,1\}^n \rightarrow \{0,1\}^m$ in the Toeplitz family is specified by a seed $S = [U, v]$, where U is a random $m \times n$ Toeplitz matrix over $GF(2)$, and $v \in \{0,1\}^m$ is a random vector. Given a vector $i \in \{0,1\}^n$, $f(S,i) = Ui + v$, where the operations are over $GF(2)$. Notice that each f in the family is specified by $n + 2m - 1$ bits, the seed.

For $m = 1$, i.e., two-valued 2-wise independent random variables, the Toeplitz scheme is identical with BCH3, thus we do not need to analyze this scheme independently.

3.8 Tabulation Based Schemes

For degrees of independence higher than three, both the BCH and the polynomials schemes require exponentiations over the extension field $GF(2^n)$, computations that are time consuming. In this case, it could be faster to pre-compute and store (tabulate) the random variables in memory, instead of generating them on the fly. When the random variable corresponding to an index i is needed, it is simply extracted from a look-up table. For large domains multiple tables are maintained, each corresponding to a different sub-domain. The random variable is obtained as a simple function of the tabulated values.

[Carter and Wegman 1979] show that the function $\bar{f}$ mapping $i_0 i_1 \dots i_{n-1}$ to $f_0[i_0] \oplus f_1[i_1] \oplus \dots \oplus f_{n-1}[i_{n-1}]$ is 3-universal if each of the independent tabulated functions f_j is 3-universal. This implies that in order to obtain a 3-universal hash function for a large sequence, it is enough to divide the sequence into sub-sequences and to tabulate 3-universal functions for each such sub-sequence. The value of the function is computed by XOR-ing the tabulated values corresponding to each sub-sequence. This result was subsequently extended to 4-universal hashing by [Thorup and Zhang 2004]. They show that the function $\bar{f}[ab] = f_0[a] \oplus f_1[b] \oplus f_2[a + b]$ is 4-universal if f_0, f_1, and f_2 are independent 4-universal hash functions.

The tabulation method reduces the processing time, but increases the required memory. As long as the the pre-computed tables fit in the fast memory, we expect an improvement in processing time. Since AMS-sketches require multiple instances of the estimator in order to provide the desired error guarantees, the space requirement could become a problem.

3.9 Performance Evaluation

We implemented BCH3, BCH5, EH3, and RM7, and we used the implementation in [Thorup and Zhang 2004] for the polynomials over primes and the tabulation schemes. BCH5 was implemented by performing the i^3 operation arithmetically

Scheme	Time (ns)	Seed size
BCH3	10.8	$n+1$
EH3	7.3	$n+1$
Polynomials2	31.4	$2n$
Tabulation2	5.1	$2^b \cdot \frac{n}{b}$
BCH5	12.7	$2n+1$
Polynomials4	106.4	$4n$
RM7	3,301	$1+n+\frac{n(n-1)}{2}$
Tabulation4	10.3	$2^b \cdot \left(\frac{n}{b}\right)^{\log_2 3}$

Table II. Generation time and seed size. For the tabulation schemes, the seed size actually represents the space occupied by the stored random variables. b represents the size (number of bits) of the stored sub-sequences.

not in an extension field. This does not change the degree of independence of BCH5 for domains not too large, but it significantly improves the performance. The polynomials scheme uses Mersenne primes in order to speed-up the implementation. For the tabulation scheme, both 8 and 16 bits sub-sequences were tested, but only the best result is reported.

We ran our experiments on a two-processor *Xeon 2.80 GHz* machine[3], each with a *512 KB* cache. The system has *1 GB* available memory and uses a *Fedora Core 3* operating system. As a comparison for our results, the time to read a word from a memory location that is not cached takes about 250 ns on this machine. We used the special assembly implementation of the dot-product for all the methods. Experiments consisted in generating $10,000$ variables i and $10,000$ seeds and then computing all possible combinations of random variables, i.e., $100,000,000$. Each experiment was run 100 times and the average of the results is reported. The relative error or error ($\textit{relative error} = \frac{|\textit{true value} - \textit{experimental value}|}{\textit{true value}}$) of a run was in general under 1%, with a maximum of 1.2%. Since the results are so stable, we do not report the actual error for individual experiments. The generation time, in nanoseconds per random variable, is reported in Table II. When compared to the memory random access time, all the schemes, except RM7, are much faster. Indeed, EH3 is at least as fast as BCH3 (we believe it is actually faster since the extra operations maintain a better flow through the processor pipelines) and both are significantly faster than the polynomials scheme. The tabulation scheme is the fastest both for 2-wise and 4-wise independent random variables since the tables fit in the cache. Notice that EH3 and BCH5 give close results to the tabulation scheme although they generate the value of the random variables from scratch.

Table II also contains the size of the seed for the given schemes. n represents the number of bits the domain I can be represented on. For the polynomials over primes scheme, n is the smallest power of 2 for which $2^n \geq p$. As noted, the BCH schemes have the densest seeds, while the Reed-Muller scheme needs the largest seed. For the same degree of independence, the polynomials over primes scheme requires a seed double in size compared to BCH. The tabulation scheme does not store seeds. Instead it stores the values of the random variables for non-overlapping sub-domains. Thus, it has the most stringent memory requirement of

[3]While the machine has two processors, only one of them was used in our experiments.

all the schemes.

When deciding what scheme to use, the tradeoffs involved should be analyzed. While the tabulation scheme is the fastest, it also requires the largest amount of memory. For AMS-sketches estimations this could become a drawback. The BCH schemes have the least restrictive memory requirement while being quite fast. We will see that for fast range-summation (efficient sketching of intervals) a *real* generating scheme gives the best results. With a tabulation scheme all the points inside an interval have to be independently sketched, a solution that we try to improve.

4. FAST RANGE-SUMMABLE GENERATION SCHEMES

In Section 2 we have seen that DMAP is the state-of-the-art solution for sketching intervals. DMAP uses dyadic mappings in order to reduce the size of an interval, thus decreasing the number of random variables that have to be generated. The fast range-summation property is the ability to compute the sum of random variables in an interval in time sub-linear in the size of the interval – the alternative is to generate and sum-up the values ξ_i for each i in the interval. This property is a characteristic of the generating scheme and it is formally defined in [Calderbank et al. 2005]:

Definition 4.1. A generating scheme for two-valued k-wise independent random variables is called **fast range-summable** if there exists a polynomial-time function g such that

$$g([\alpha, \beta], S) = \sum_{\alpha \leq i \leq \beta} \xi_i(S) = \sum_{\alpha \leq i \leq \beta} (-1)^{f(S,i)}$$

where α, β, and i are values in the domain I, with $\alpha \leq \beta$.

Computing the function g over general $[\alpha, \beta]$ intervals is usually not straightforward. The task is simpler for dyadic intervals (see Section 2.4.1) due to their regularity. Fortunately, any scheme that is fast range-summable for dyadic intervals can be extended to general intervals $[\alpha, \beta]$ by simply determining the minimal dyadic cover, computing the function g over each dyadic interval in the cover, and then summing-up these results. Since the decomposition of any $[\alpha, \beta]$ interval contains at most a logarithmic number of dyadic intervals, fast range-summable algorithms for dyadic intervals remain fast range-summable for general intervals. Notice how dyadic intervals are used in different ways for fast range-summation and in DMAP. While DMAP represents an interval by its minimal dyadic cover and random variables are generated for each dyadic interval in the cover, dyadic intervals speed-up the summing of the random variables in fast range-summation methods.

In this section, we study the fast range-summation property of the generating schemes presented in Section 3. For 2-wise independent random variables, both the BCH3 scheme and its EH3 variant are fast range-summable. [Bar-Yosseff et al. 2002] show that the Toeplitz family of hash functions is fast range-summable. Since for two-valued random variables the Toeplitz scheme is equivalent with BCH3, the algorithm we introduce for BCH3 also applies for the Toeplitz scheme. A related algorithm for p-valued 2-wise independent random variables generated using the polynomials over primes scheme is introduced in [Aduri and Tirthapura 2005]. For

the 4-wise case, the Reed-Muller generating scheme is the only scheme known to be fast range-summable [Calderbank et al. 2005; Gilbert et al. 2003].

As suggested in [Levchenko 2005], reducing the range-summing problem to determining the number of boolean variables assignments that satisfy an XOR-AND logical expression, and using the results in [Ehrenfeucht and Karpinski 1990], is a formal method to determine if a generating scheme is fast range-summable. We apply this method to show that neither BCH5 nor the polynomial schemes are fast range-summable.

4.1 BCH3 Scheme

We prove that BCH3 is fast range-summable and we provide an average case constant time $O(1)$ algorithm for computing the sum of the ± 1 random variables in any $[\alpha, \beta]$ interval. The BCH3 generating function f has the following form:

$$f(S,i) = [s_0, S_0] \cdot [1, i] = s_0 \oplus \bigoplus_{k=0}^{n-1} S_{0,k} \odot i_k \tag{7}$$

where $\oplus$ denotes the bitwise XOR, and $\odot$ denotes the bitwise AND. Both S_0 and i are vectors over the space $GF(2)^n$. Given two vectors in $GF(2)^n$, α and β, with $\alpha \leq \beta$, when interpreted as binary numbers, the problem of fast range-summing function f over the interval $[\alpha, \beta]$ is to compute the function g defined below:

$$\begin{aligned} g([\alpha,\beta],S) &= \sum_{\alpha \leq i \leq \beta} (-1)^{f(S,i)} \\ &= \sum_{\alpha \leq i \leq \beta} (-1)^{[s_0,S_0]\cdot[1,i]} = \sum_{\alpha \leq i \leq \beta} (-1)^{s_0 \oplus \bigoplus_{k=0}^{n-1} S_{0,k} \odot i_k} \end{aligned} \tag{8}$$

Notice that the operations that involve the seed and the index variable are over $GF(2)$, but the summation over the interval $[\alpha, \beta]$ is in $\mathbb{Z}$. Initially, we consider that the interval $[\alpha, \beta]$ is dyadic. This means that it has the form $[q2^j, (q+1)2^j)$, with $0 \leq j \leq n$ and $0 \leq q \leq 2^{n-j} - 1$. Lemma 4.4 shows how to evaluate the function g for the BCH3 scheme over a dyadic interval, but we first introduce some results that are useful for the subsequent proofs.

PROPOSITION 4.2. *Consider the function XOR*$(x_1, x_2, \ldots, x_n) = x_1 \oplus x_2 \oplus \cdots \oplus x_n$ *defined over* n *binary variables* $x_1, x_2, \ldots, x_n$. *Then, the function XOR takes the values* 0 *and* 1 *equally often, each value* 2^{n-1} *times.*

PROOF. We prove this proposition by induction on n.

Base case: $n = 2$. The truth table for the XOR function with two variables is given in Table III. Function XOR takes the values 0 and 1 the same number of times, $2^{2-1} = 2$.

Inductive case: We suppose that the proposition is true for n and we have to show that it is also true for $n+1$, that is, function $x_1 \oplus x_2 \oplus \cdots \oplus x_n \oplus x_{n+1}$ takes both values 0 and 1 2^n times each. For every combination of the first n variables, x_{n+1} takes one time the value 0 and one time the value 1. When $x_{n+1} = 0$, function XOR of $n+1$ variables is identical with function XOR of n variables. This means that it takes the values 0 and 1 2^{n-1} times each. The effect of x_{n+1} being equal with 1 is to invert the result of the function XOR with n variables, that is, the

x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Table III. Truth table for the function $x_1 \oplus x_2$.

combinations that gave result 0, give 1 now, and the combinations that gave 1 for n variables, give 0 for $n+1$ variables. As we already know that the number of combinations for both cases is equal to 2^{n-1} times each, by summing the results for $x_{n+1} = 0$ and $x_{n+1} = 1$, we obtain that function XOR with $n+1$ variables takes the values 0 and 1 2^n times each. □

PROPOSITION 4.3. *Let the function F on n binary variables $x_1, x_2, \ldots, x_n$ be defined as:*

$$F(x_1, x_2, \ldots, x_n) = C \oplus S_1 \odot x_1 \oplus S_2 \odot x_2 \oplus \cdots \oplus S_n \odot x_n$$

where $C \in \{0,1\}$ is a constant and $S_k \in \{0,1\}$, for $1 \leq k \leq n$, are parameters. If there exists at least one S_k that is equal to 1, function F takes the values 0 and 1 equally often, each value 2^{n-1} times. Otherwise, function F evaluates to the constant C for all the combinations of $x_1, x_2, \ldots, x_n$.

PROOF. Let $S_{k_1}, S_{k_2}, \ldots, S_{k_r}$, $0 \leq r < n$, be the parameters that are constrained to the value 0 and $S_{k_{r+1}}, \ldots, S_{k_n}$ be the parameters that equal 1. Then, we can rewrite function F as:

$$F(x_1, x_2, \ldots, x_n) = C \oplus x_{k_{r+1}} \oplus \cdots \oplus x_{k_n}$$

When $C = 0$ it has no effect on the value of the function F, while $C = 1$ inverts the result of the function. We know from Proposition 4.2 that the function XOR with $n-r$ variables takes the values 0 and 1 2^{n-r-1} times each. The 2^n combinations of n variables consist now of 2^r repetitions of the 2^{n-r} combinations of $n-r$ variables. It results that function F takes the value 0, as well as value 1, $2^{n-r-1} \cdot 2^r = 2^{n-1}$ times, for $C = 0$. The same result holds for $C = 1$ since $2^n - 2^{n-1} = 2^{n-1}$. For $r = n$, we have $F(x_1, x_2, \ldots, x_n) = C$. □

LEMMA 4.4. *Let $[q2^j, (q+1)2^j)$ be a dyadic interval with $1 \leq j \leq n$ and $0 \leq q \leq 2^{n-j} - 1$, and let function g be defined as:*

$$g([q2^j, (q+1)2^j), S) = \sum_{i=q2^j}^{(q+1)2^j - 1} (-1)^{f(S,i)}$$

where $S = [S_0, s_0]$, with S_0 a vector in $GF(2)^n$ and $s_0 \in \{0,1\}$, and function f is defined as $f([S_0, s_0], i) = s_0 \oplus \bigoplus_{k=0}^{n-1} (S_{0,k} \odot i_k)$. Then, function g can take the following values:

$$g([q2^j, (q+1)2^j), [S_0, s_0]) = \begin{cases} 0 & \text{, if } 2^j \text{ does not divide } S_0 \\ 2^j \cdot (-1)^{f([S_0,s_0], q2^j)} & \text{, if } 2^j \text{ divides } S_0 \end{cases}$$

PROOF. Function f can be rewritten as follows for the interval $[q2^j, (q+1)2^j)$:

$$\begin{aligned} f([S_0, s_0], i) &= s_0 \oplus \bigoplus_{k=j}^{n-1} (S_{0,k} \odot i_k) \oplus \bigoplus_{k=0}^{j-1} (S_{0,k} \odot i_k) \\ &= C \oplus \bigoplus_{k=0}^{j-1} (S_{0,k} \odot i_k) \end{aligned}$$

where $C \in \{0, 1\}$ is constant for a given S_0. This was possible because the most significant $n - j$ bits of a $[q2^j, (q+1)2^j)$ dyadic interval are identical for all the values in the interval. Function g becomes:

$$g([q2^j, (q+1)2^j), S) = \sum_{i=q2^j}^{(q+1)2^j-1} (-1)^{C \oplus \bigoplus_{k=0}^{j-1}(S_{0,k} \odot i_k)}$$

Applying Proposition 4.3 for the expression $C \oplus \bigoplus_{k=0}^{j-1}(S_{0,k} \odot i_k)$ (provided that $S_{0,k} \neq 0$, $0 \leq k < j$), we know that the values 0 and 1 are taken equally often, thus the sum in function g gives the result 0. When all the j least significant bits of S_0 are equal to 0, that is, 2^j divides S_0, function f equals the constant C and function g can be written as:

$$\begin{aligned} g([q2^j, (q+1)2^j), S) &= \sum_{i=q2^j}^{(q+1)2^j-1} (-1)^C \\ &= 2^j \cdot (-1)^C \end{aligned}$$

where $C = s_0 \oplus \bigoplus_{k=j}^{n-1}(S_{0,k} \odot i_k)$. In order to evaluate C and function g, it is enough to compute function f for any of the points in the dyadic interval, for example $q2^j$. □

Lemma 4.4 gives a method to compute the sum of the BCH3 random variables for any dyadic interval $[q2^j, (q+1)2^j)$ in a constant number of operations. Notice that, for the majority of cases, there is no need to generate even one random variable in the interval to find the result of the summation. Only when S_0 is a multiple of 2^j a computation of the function f is required in order to compute the sum. The value of the sum in this particular case is either -2^j or 2^j.

Thus far, we have seen that it is possible to fast range-sum the BCH3 random variables over a dyadic interval by generating at most one of the variables in the interval. The generation is required only when the least significant j bits in S_0 are all equal to 0. When this is not the case, i.e., there is at least one bit equal to 1, we know the result of the summation without computing anything. Verifying that the least significant j bits in the binary representation of a value are all equal to 0 can be done efficiently on any processor, e.g., AND-ing with $2^{j+1} - 1$. We conclude that, in order to fast range-sum BCH3 random variables over a dyadic interval $[q2^j, (q+1)2^j)$, it is enough to analyze the values of the least significant j bits in S_0 and, sometimes, to generate one of the variables in the interval. Both these tasks can be carried out efficiently.

The general fast range-summing problem is to compute the function g defined in (8) for any interval $[\alpha, \beta]$, $\alpha \leq \beta$. As we know from Section 2.4.1, any interval can be represented as a union of dyadic intervals. By decomposing the interval $[\alpha, \beta]$ into its minimal dyadic cover $D([\alpha, \beta]) = \{\delta_1, \delta_2, \ldots, \delta_m\}$, function g can be rewritten as:

$$\begin{aligned} g([\alpha,\beta],S) &= \sum_{\alpha \leq i \leq \beta} (-1)^{f(S,i)} = \sum_{i \in \cup_{k=1}^{m} \delta_k} (-1)^{f(S,i)} \\ &= \sum_{i \in \delta_1} (-1)^{f(S,i)} + \sum_{i \in \delta_2} (-1)^{f(S,i)} + \cdots + \sum_{i \in \delta_m} (-1)^{f(S,i)} \end{aligned} \tag{9}$$

where δ_k, $1 \leq k \leq m$, are dyadic intervals. Algorithm *MinimalDyadicCover* computes the minimal dyadic cover of an interval $[\alpha, \beta]$, while Lemma 4.4 shows how to efficiently compute each of the sums in (9). The straightforward implementation of these two steps results into a fast range-summable algorithm for the BCH3 generating scheme since the number of intervals in the minimal dyadic cover is logarithmic in the size of the interval and the computation of the sum over each of these intervals needs constant time. We go one step further and introduce an algorithm that overlaps these two steps, decomposition and summation. Besides overlapping these two stages, the algorithm we propose has strong early termination conditions that make it a constant time algorithm for the average case (over the seed space), improving considerably over the existing logarithmic fast range-summable algorithms.

BCH3Interval($[0, \gamma], S = [S_0, s_0]$)

```
1  γ' ← γ
2  sum ← 0
3  if γ'_0 = 1
4     then γ' ← γ' −1
5          sum ← (−1)^f(S,γ')
6  for k ← 1 to n −1
7        do
8           if γ' = 0
9              then return sum
10          if S_0,k-1 = 1
11             then return sum
12             else  ▷ S_0,k-1 = 0
13                   if γ'_k = 1
14                      then γ' ← γ' −2^k
15                           sum ← sum +2^k · (−1)^f(S,γ')
16 return sum
```

Instead of computing function g directly over the interval $[\alpha, \beta]$, we write it as a difference over the intervals $[0, \beta + 1)$ and $[0, \alpha)$, $g([\alpha, \beta], S) = g([0, \beta + 1), S) - g([0, \alpha), S)$. The advantage of this form comes from the fact that the minimal dyadic cover of a $[0, \gamma)$ interval, $\gamma \in GF(2)^n$, can be directly determined from the binary representation of γ. If $\gamma = \gamma_{n-1}\gamma_{n-2}\ldots\gamma_0$, $\gamma_k \in \{0, 1\}$ for $0 \leq k < n$, the minimal dyadic cover of the interval $[0, \gamma)$ contains a size 2^k interval for every γ_k equal with 1. Based on this observation and on Lemma 4.4, Algorithm *BCH3Interval*

computes function g over any $[0, \gamma)$ interval.

Lines $1-2$ initialize the variables *sum* and γ'. *sum* stores the result of function g. γ' is just a substitute of γ that is modified inside the algorithm. When γ is odd, that is, $D([0, \gamma)$ contains a point interval, function f is computed separately for it (Lines $3-5$). The for-loop in the Lines $6-15$ is the core of the algorithm. It detects the dyadic intervals in $D([0, \gamma))$ and computes the sum of the random variables inside them according to Lemma 4.4.

There exist two exit points from the algorithm inside the for-loop. The first one (Lines $8-9$) is straightforward: the algorithm returns when the range-sum is computed over the entire interval. In this case γ' equals 0 since it is decremented after the detection of each interval in the minimal dyadic cover. The second exit point in the for-loop (Lines $10-11$) is more interesting. We know from Lemma 4.4 that the range-sum is determined without any computation for dyadic intervals that have non-zero corresponding bits in the seed S_0. We iterate through the intervals in the dyadic cover in ascending order of their size and the existence of a single one bit in S_0 automatically determines the range-sum for all subsequent intervals in the cover. This enables us to compute the value of function g without generating other random variables (we know that the sum is 0 in this case).

Lines $12-15$ correspond to the case when the least significant k bits in S_0 are all equal to 0. As shown in Lemma 4.4, a random variable inside the dyadic interval has to be generated in this case. The subtraction in Line 14 imposes that the random variable corresponding to the first point in the dyadic interval is generated. Line 16 contains the last exit point. It can be attained only when the seed S_0 is equal to 0 and γ_{n-1} equals to 1. Notice that the operations that appear in the algorithm imply computations with 2 and its powers and that they can be efficiently implemented using bit operations (AND, XOR, SHIFT, etc.).

Algorithm *BCH3* shows how to correctly invoke *BCH3Interval* for $[\alpha, \beta]$ intervals. Notice that the first call to *BCH3Interval* is with $\beta + 1$. This is necessary for the correct computation of function g.

BCH3($[\alpha, \beta], S = [S_0, s_0]$)
1 $sum_1 \leftarrow$ BCH3Interval($[0, \beta + 1), \mathrm{S}$)
2 $sum_2 \leftarrow$ BCH3Interval($[0, \alpha), \mathrm{S}$)
3 **return** $sum_1 - sum_2$

Example 4.5. *We show how Algorithm BCH3 works for the interval* $[100, 202]$ *and the seeds* $s_0 = 0$, $S_0 = 184 = (10111000)_2$. sum_1 *is computed over the interval* $[0, 203)$, *while* sum_2 *is calculated over the interval* $[0, 100)$. *Their difference is returned. For* $203 = (11001011)_2$, *function* f *is invoked first outside the for-loop, with* $\gamma' = 202 = (11001010)_2$, sum_1 *taking the value* $(-1)^0 = 1$. f *is then invoked for* $\gamma' = 200 = (11001000)_2$, *giving the result* $2 \cdot (-1)^0 = 2$. sum_1 *is updated to* 3. *The last time* f *is invoked for the interval* $[0, 203)$, $\gamma' = 192 = (11000000)_2$ *and the returned result is* $2^3 \cdot (-1)^1 = -8$, *the value of* sum_1 *until this point being* -5. *At the next iteration* $k = 4$ *and* $S_{0,3} = 1$ *and the routine BCH3Interval returns the value* -5 *for* sum_1. *For* $100 = (01100100)_2$ *function* f *is called only once, with* $\gamma' = 96 = (01100000)_2$, sum_2 *taking the value* $2^2 \cdot (-1)^1 = -4$. *At the fourth iteration of the for-loop,* -4 *is returned for* sum_2. *Finally, the range-sum of the*

BCH3 function for the interval $[100, 202]$ *is* $-5-(-4) = -1$.

THEOREM 4.6. *Let* $[\alpha, \beta]$, $\alpha \leq \beta$, *be an interval, where* $\alpha, \beta \in GF(2)^n$, *and seed* $S = [S_0, s_0]$ *be a vector over* $GF(2)^{n+1}$, *i.e.,* $S_0 \in GF(2)^n$ *and* $s_0 \in \{0, 1\}$. *Then, Algorithm BCH3 computes the sum of the BCH3 random variables over the interval* $[\alpha, \beta]$.

PROOF. Instead of range-summing over the interval $[\alpha, \beta]$, we apply Algorithm *BCH3Interval* over the intervals $[0, \beta + 1)$ and $[0, \alpha)$ and return the difference of these results.

Algorithm *BCH3Interval* has two termination points: exhausting the entire interval $[0, \gamma)$ and detecting the first least significant one bit in S_0. For every interval that is part of the minimal dyadic cover $D([0, \gamma))$, γ is decreased accordingly. When the entire minimal dyadic cover is determined, γ equals 0 and the algorithm returns. This is the ordinary termination, for which the algorithm executes the for-loop a logarithmic number of times in the size of the $[0, \gamma)$ interval. A reduction to a constant number of executions of the for-loop is provided by the second return point, the detection of the first one bit in S_0. We know from Lemma 4.4 that the range-sum over dyadic intervals that depend on non-zero seeds S_0 can be computed without generating any random variable in the interval. Applying this result to the current $[0, \gamma)$ interval that is a union of dyadic intervals that depend on a non-zero seed S_0, the range-sum can be immediately returned.

Algorithm *BCH3Interval* returns the range-sum over $[0, \gamma)$ intervals. The minimal dyadic cover $D([0, \gamma))$ is encoded in the binary representation of γ and consists of the intervals that have decreasing sizes as we go from 0 toward γ. That is, for $D([0, \gamma)) = \{\delta_1, \delta_2, \ldots, \delta_{\gamma_m}\}$, $|\delta_1| > |\delta_2| > \cdots > |\delta_{\gamma_m}|$ holds. The range-sum over each δ_k dyadic interval, $1 \leq k \leq \gamma_m$, is computed according to Lemma 4.4. Since the intervals are detected in the increasing order of their sizes, it is possible to determine the value of the range-sum without any other computation, when the fast termination condition is met: the partial sum is 0, as stated in Lemma 4.4. □

THEOREM 4.7. *Algorithm BCH3Interval performs on expectation* 2 *computations of the function* f, *one outside the loop and one inside the for-loop, given that the seeds* $S = [S_0, s_0]$ *are* 2*-wise independent and uniformly distributed over* $GF(2)^{n+1}$. *For* 100 *seeds* S, *the average number of function* f *computations is between* 1.723 *and* 2.277, *while for* $1,000$ *seeds it is inside the interval* $[1.912, 2.088]$. *For* $10,000$ *seeds it lies between* 1.972 *and* 2.028. *All these results have a* 0.95 *confidence probability.*

PROOF. We consider that γ has the worst value for our algorithm, e.g., $\gamma = \overbrace{1\ldots11}^{n}$. This way, it always generates a computation of the function f. We prove that the fast termination condition makes the algorithm to execute the for-loop 1 time, on expectation, giving a total of 2 function f computations.

For this, we have to determine the average number of 0 consecutive bits that appear in the least significant positions of S_0, knowing that S_0 is uniformly distributed over the space $GF(2)^n$. Since the bits in S_0 are independent, the probability of having exactly k consecutive bits with value 0, $1 \leq k \leq n$, preceded by a 1, is $\frac{1}{2^{k+1}}$. We define the random variable X as the number of least significant consecutive 0

bits in S_0 over the uniform probability space $GF(2)^n$. The expected value of X, $E[X]$, gives us exactly what we are looking for, that is, the average number of least significant consecutive 0 bits in S_0.

$$\begin{aligned} E[X] &= 1 \cdot \frac{1}{2^2} + 2 \cdot \frac{1}{2^3} + \cdots + n \cdot \frac{1}{2^{n+1}} \\ &= \frac{1}{2} \sum_{k=1}^{n} k \cdot \left(\frac{1}{2}\right)^k \lessapprox 1 \end{aligned}$$

$E[X]$ is a derived power series with ratio $\frac{1}{2}$ that converges to 1. For a complete characterization of the random variable X, we also compute its variance, $Var[X]$, which tells us what is the range around the expected value variable X takes values in.

$$Var[X] = E[X^2] - E^2[X]$$

$$\begin{aligned} E[X^2] &= 1^2 \cdot \frac{1}{2^2} + 2^2 \cdot \frac{1}{2^3} + \cdots + n^2 \cdot \frac{1}{2^{n+1}} \\ &= \frac{1}{2} \sum_{k=1}^{n} k^2 \cdot \left(\frac{1}{2}\right)^k \lessapprox 3 \end{aligned}$$

$$\begin{aligned} Var[X] &= E[X^2] - E^2[X] \\ &= 3 - 1^2 \lessapprox 2 \end{aligned}$$

Both the expectation and the variance of X have finite values, e.g., 1 and, respectively, 2. For a large enough number of seeds S, e.g., greater than 100, the central limit theorem [Shao 1999] can be applied. Let the random variable Y be defined as:

$$Y = \frac{\sum_{k=1}^{N} X_k}{N}$$

where N is the number of seeds S, $N \geq 100$. Using the central limit theorem, we know that random variable Y can be approximated by a normal distribution with parameters $E[Y] = E[X]$ and $Var[Y] = \frac{Var[X]}{N}$. We determine the range of variable Y within a 0.95 confidence interval using the cumulative distribution function (cdf) of the normal distribution that approximates Y.

$$Y_l = 1 + \frac{2}{\sqrt{N}} \cdot \mathrm{Erf}^{-1}(-0.95)$$

$$Y_r = 1 + \frac{2}{\sqrt{N}} \cdot \mathrm{Erf}^{-1}(0.95)$$

Y_l and Y_r are the left and right interval end-points, while Erf^{-1} is the inverse of the error function [Shao 1999]. Replacing N with 100, 1,000 and 10,000 in the above formulae, and adding 1 for the out of loop function f computation, we obtain the results stated in the theorem. □

In Theorem 4.7 we considered that γ takes the worst value for our algorithm, e.g., $\gamma = \overbrace{1 \ldots 11}^{n}$. In practice, this does not happen too often and the number of function f computations could be smaller. If we assume that the bits in γ take the values 0 and 1 with the same probability, the number of computations of f reduces to half, i.e., 1. Also, there exist cases when no computation of f is done, e.g., $\gamma_0 = 0$ and $S_{0,0} = 1$, i.e., the least significant bits in γ and S_0 are 0 and, respectively, 1.

COROLLARY 4.8. *Function f is computed on expectation 4 times when BCH3 random variables are range-summed over $[\alpha, \beta]$ intervals.*

PROOF. Algorithm *BCH3Interval* is called two times by Algorithm *BCH* and each invocation computes function f 2 times on expectation. □

The result in Corollary 4.8 is for interval end-points α and β that are worst for our algorithm, e.g., both α and $\beta + 1$ end in 11. In the average case, with the bits taking the values 0 and 1 uniformly probable, the number of function f computations reduces to 2. This result is the best we could hope for, that is, computing the range-sum over an interval $[\alpha, \beta]$ by taking into consideration only its end-points, α and β.

4.2 EH3 Scheme

Although [Feigenbaum et al. 2002] show that the random variables generated using the Extended Hamming scheme (EH3) are fast range-summable, the algorithm contained in the proof is abstract and not appropriate for implementation purposes. We propose a practical algorithm for the fast range-summation of the EH3 random variables. It is an extension of our constant-time algorithm for fast range-summing BCH3 random variables.

The following theorem provides an analytical formula for computing the range-sum function g. Notice that only one computation of the generating function f is required in order to determine the value of g over any dyadic interval.

THEOREM 4.9. *Let $[q4^j, (q+1)4^j)$ be a dyadic interval[4] with size at least 4, $j \geq 1$. The range-sum function $g([q4^j, (q+1)4^j), S) = \sum_{i=q4^j}^{(q+1)4^j} (-1)^{f(S,i)}$ defined for EH3 scheme is equal to:*

$$g([q4^j, (q+1)4^j), S) = (-1)^{\#ZERO} \cdot 2^j \cdot (-1)^{f(S,q4^j)} \tag{10}$$

where #ZERO represents the number of two adjacent pair bits that OR to 0 (i.e., the number of groups of 00 bits).

PROOF. The generating function f can be written for $i \in [q4^j, (q+1)4^j)$ as follows:

$$\begin{aligned} f(S,i) = f(S, q4^j) \oplus \\ S_{0,2j-1} \odot i_{2j-1} \oplus S_{0,2j-2} \odot i_{2j-2} \oplus i_{2j-1} \odot i_{2j-2} \oplus \cdots \oplus \\ S_{0,1} \odot i_1 \oplus S_{0,0} \odot i_0 \oplus i_1 \odot i_0 \end{aligned}$$

[4]Although we still call them dyadic intervals, these intervals are defined over powers of 4.

where the first part is fixed, while the last $2j$ bits of i take all the possible values. When the value of the changing expression is 0, $f(S,i) = f(S, q4^j)$, while when its value is 1, $f(S,i)$ is the negation of $f(S, q4^j)$. We know that any expression of the form $S_{0,j} \odot i_j \oplus S_{0,j-1} \odot i_{j-1} \oplus i_j \odot i_{j-1}$ has a $3:1$ distribution of the values, depending on the seed bits. Thus, the sum $\sum(S_{0,j} \odot i_j \oplus S_{0,j-1} \odot i_{j-1} \oplus i_j \odot i_{j-1})$ takes either the value 2 or -2 when i_j and i_{j-1} take all the possible values. When j blocks of this form are combined together, the resulting sum will be 2^j or -2^j. For one block, the sum is 2 only when $S_{0,j} \vee S_{0,j-1} = 0$, i.e., $S_{0,j} = S_{0,j-1} = 0$. For multiple blocks that are XOR-ed, the result is 1 only when an odd number of them takes the value 1. This implies that in order to obtain the result 2^j, $S_{0,j} = S_{0,j-1} = 0$ has to be valid for an odd number of blocks. If we denote by #ZERO the number of blocks for which the relation $S_{0,j} = S_{0,j-1} = 0$ holds, the range-sum function g can be written as:

$$g([q4^j, (q+1)4^j), S) = (-1)^{\#\text{ZERO}} \cdot 2^j \cdot (-1)^{f(S,q4^j)}$$

□

Based on the results in Theorem 4.9, Algorithm *EH3Interval* computes function $g([\alpha,\beta],S) = \sum_{\alpha \le i \le \beta}(-1)^{f(S,i)}$ for any interval $[\alpha,\beta]$. First, the minimal dyadic cover of $[\alpha,\beta]$ is determined, then the sum over each dyadic interval is computed using (10). Notice that these two steps can be combined, the computation of g being performed while determining the minimal dyadic cover of $[\alpha,\beta]$. The minimal dyadic cover can be efficiently determined from the binary representation of α and β. Since any interval can be decomposed into a logarithmic number of dyadic intervals, algorithm *EH3Interval* computes function g in $O(\log(\beta-\alpha))$ steps.

EH3Interval$([\alpha,\beta], S = [S_0, s_0])$
1 Let $D = \{\delta_1, \ldots, \delta_m\}$ be the minimal dyadic cover of $[\alpha,\beta]$
2 $sum \leftarrow 0$
3 **for** $\delta \in D$
4 **do** $sum \leftarrow sum + (-1)^{\#\text{ZERO}} \cdot 2^j \cdot (-1)^{f(S,q4^j)}$
5 **return** *sum*

Example 4.10. *We show how Algorithm* EH3Interval *works for the interval* $[124, 197]$ *and the seed* $S = [s_0, S_0] = [0, 184 = (10111000)_2]$. *The minimal dyadic cover of* $[124, 197]$ *is*

$$D([124,197]) = \{[124,128), [128,192), [192,196), [196,197), [197,198)\}$$

#ZERO is equal with 1 *for the given* S_0, *the only pair OR-ing to* 0 *being the pair at the end. It affects the dyadic intervals with powers greater than* 0.

$$\begin{aligned} g([124,197],S) &= g([124,128),S) + g([128,192),S) + g([192,196),S) + \\ &\quad g([196,197),S) + g([197,198),S) \\ &= -2^1 \cdot (-1)^{f(S,124)} - 2^3 \cdot (-1)^{f(S,128)} - 2^1 \cdot (-1)^{f(S,192)} + \\ &\quad 2^0 \cdot (-1)^{f(S,196)} + 2^0 \cdot (-1)^{f(S,197)} \\ &= 2 + 8 + 2 + 1 - 1 = 12 \end{aligned}$$

While fast range-summing BCH3 random variables is constant-time on average, the EH3 algorithm is logarithmic in the size of the interval. Although both algorithms compute partial sums over the dyadic intervals in the minimal dyadic cover, the BCH3 algorithm can return the final result after only a small number of function f invocations. This is not the case for EH3, which requires one invocation for each dyadic interval in the minimal dyadic cover.

4.3 Four-Wise Independent Schemes

In this section we investigate the fast range-summation property of the 4-wise independent generating schemes presented throughout this paper, namely BCH and polynomials over primes. The discussion regarding the Reed-Muller scheme is deferred to the next section.

The main idea in showing that some of the schemes are not fast range-summable is to use the result in [Ehrenfeucht and Karpinski 1990] on the problem of counting the number of times a polynomial over $GF(2)$, written as XOR of ANDs (sums of products with operations in $GF(2)$), takes each of the two values in $GF(2)$, as suggested by [Levchenko 2005]. The result states that the problem is #P-complete if any of the terms of the polynomial written as an XOR of ANDs contains at least three variables. In our particular case, to show that a scheme is not fast range-summable, it is enough to prove that for some seed S the generating function $f(S, i)$, written as an XOR of ANDs polynomial in the bits of i, contains at least one term that involves three or more variables. The following results use this fact to show that the BCH5 and the polynomials over primes schemes are not fast range-summable.

THEOREM 4.11. *BCH schemes are not fast range-summable for $k \geq 5$ and $n \geq 4$.*

PROOF. We show that fast range-summing BCH5 random variables is equivalent to determining the number of assignments that satisfy a 3XOR-AND boolean formula, problem that is known to be #P-complete. Since for $k > 5$ any BCH scheme can be reduced to BCH5 for some particular values of the seed, i.e., $S_2 = S_3 = \cdots = S_{k/2} = \bar{0}$, if BCH5 is not fast range-summable implies that BCHk is not fast range-summable, for $k > 5$.

BCH5 necessitates the computation of i^3 over the extension field $GF(2^n)$. If we consider the bit representation of i, it can be shown that i^3 is a 3XOR-AND boolean formula for $n > 3$ (the idea is to use the bit equations of a multiplier). This reduction implies that BCH5 is not fast range-summable. □

THEOREM 4.12. *Let $n = \lceil \log p \rceil$ be the number of bits the prime $p > 7$ can be represented on and $[q2^l, (q+1)2^l)$ be a dyadic interval with $l \geq 3$. Then, the function $f(S, i) = [(a_0 + a_1 i) \mod p] \mod 2$ is not fast range-summable over the interval $[q2^l, (q+1)2^l)$.*

PROOF. Without loss of generality, we consider that $a_0 = 0$, $a_1 = p - 1$, and show that function g is a 3XOR-AND formula, implying that it is not fast range-summable. The expression $(p-1)\bar{i} \mod p$ takes the values $0, p-1, p-2, \ldots, p-7$ for $\bar{i} = 0, 1, \ldots, 7$. Out of these values, only $p-2, p-4,$ and $p-6$ are odd, function g taking the value 3. If we express the result of function g as an XOR-AND formula

in terms of $i_0, i_1,$ and i_2, we obtain:

$$g(i_2 i_1 i_0, S) = i_1 \oplus i_2 \oplus i_1 \odot i_0 \oplus i_2 \odot i_0 \oplus i_2 \odot i_1 \oplus i_2 \odot i_1 \odot i_0$$

which is a 3XOR-AND formula. We know that the number of assignments to i_2, i_1, i_0 that satisfy this formula cannot be determined in polynomial time. This implies that function f is not fast range-summable over dyadic intervals of size at least 8. □

Theorem 4.12 shows that for the polynomials over primes scheme with $k = 2$ there exist values for the coefficients a_0 and a_1 that make the scheme not fast range-summable for dyadic intervals with size greater or equal than $2^3 = 8$. Since the schemes for $k > 2$ can be reduced to $[(a_0 + a_1 i) \mod p] \mod 2$ by making $a_2 = \cdots = a_{k-1} = 0$, it results that the polynomials over primes scheme is not fast range-summable for any $k \geq 2$.

4.4 RM7 Scheme

Together with the negative result on the hardness of counting the number of times an XOR of ANDs polynomial with terms containing more that three variables AND-ed, [Ehrenfeucht and Karpinski 1990] provided an algorithm for such counting for formulae that contain only at most two variables AND-ed in each term. This algorithm, that we refer to as 2XOR-AND, can be readily used to produce a fast range-summable algorithm for the 7-wise independent Reed-Muller (RM7) scheme. An algorithm for this scheme based on the same ideas was proposed in [Calderbank et al. 2005]. We focus our discussion on the 2XOR-AND algorithm, but the same conclusions are applicable to the algorithm in [Calderbank et al. 2005].

The observation at the core of the 2XOR-AND algorithm is the fact that polynomials with a special shape are fast range-summable. These are polynomials with at most two variables AND-ed in any term and with each variable participating in at most one such term. The other cases can be reduced to this case by introducing new variables that are linear combinations of the existing ones. To determine these linear combinations in the general case, systems of linear equations have to be constructed and solved, one for each variable. The overall algorithm is $O(n^3)$, with n the number of variables, if the summation is performed over a dyadic interval.

The 2XOR-AND algorithm can be used to fast range-sum random variables produced by the RM7 scheme since in the XOR of ANDs representation of this scheme as a polynomial of the bits of i (which is the representation used in Section 3) only terms with ANDs of at most two variables appear. Using the 2XOR-AND algorithm for each dyadic interval in the minimal dyadic cover of a given interval, the overall running time can be shown to be $O(n^4)$ where the size of I, the domain, is 2^n. While this algorithm is clearly fast range-summable using the definition, in practice it might still be too slow to be useful. Indeed this is the case, as it is shown in the empirical evaluation section where we provide a running time comparison of the fast range-summable algorithms.

4.5 Approximate Four-Wise Independent Schemes

Since the RM7 fast range-summable algorithm is not practical and fast range-summable algorithms for BCH5 and polynomials over primes schemes do not exist,

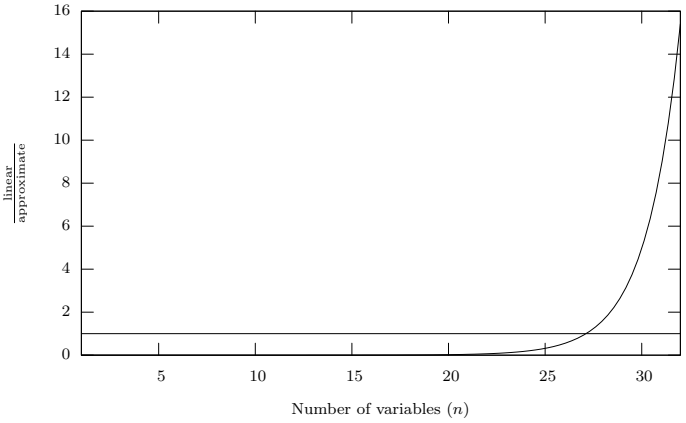

Fig. 4. The number of polynomial evaluations $\frac{\text{linear}}{\text{approximate}}$.

it is worth investigating approximation algorithms for the 4-wise case. While such approximations are possible [Karpinski and Luby 1993; Luby and Wigderson 1995], we show that they are not more practical than the exact algorithm for RM7.

Let *xaf* be a multivariate polynomial in the variables $x_1, x_2, \ldots, x_n$ over $GF(2)$ and having the form:

$$xaf(x_1, x_2, \ldots, x_n) = t_1(x_1, x_2, \ldots, x_n) \oplus \cdots \oplus t_m(x_1, x_2, \ldots, x_n)$$

where for each $j = 1, 2, \ldots, m$, the term $t_j(x_1, x_2, \ldots, x_n)$ is the product of a subset of the variables $x_1, x_2, \ldots, x_n$. The approximate XOR-AND counting algorithm introduced in [Karpinski and Luby 1993] needs $4m^2 \ln(\frac{2}{\delta})\frac{1}{\epsilon^2}$ random trials – particular value assignments to all of the variables – to provide a result with relative error at most ϵ with probability at least $1 - \delta$. Each of these trials necessitates the evaluation of the polynomial xaf at the particular assigned value. In order to have an efficient approximate algorithm, the number of trials it evaluates should be small. We show that for obtaining a good approximation of the range-sum problem, the number of trials is comparable with the size of the interval even for the RM7 scheme, thus making the algorithm impractical.

Figure 4 represents the number of evaluations of the polynomial xaf for the RM7 scheme. The ratio between the number of linear point-by-point evaluations and the number of evaluations invoked by the approximate algorithm in [Karpinski and Luby 1993] is plotted for a number of variables that ranges between 1 and 32. The number of terms m in the RM7 polynomial is $\frac{n(n+1)}{2}$. In the average case, half of these terms are simplified by corresponding 0 seeds, giving a total of $\frac{n(n+1)}{4}$ terms. In order to obtain a satisfactory approximation of the XOR-AND counting result, we set the value of the factor $\ln(\frac{2}{\delta})\frac{1}{\epsilon^2}$ to 10^3. All these give the value $10^3 \cdot \frac{n^2(n+1)^2}{4}$ for the number of trials employed by the approximate XOR-AND counting algorithm. As the number of polynomial evaluations for the linear case is 2^n, the function plotted in Figure 4 is $\frac{2^n}{10^3 \cdot \frac{n^2(n+1)^2}{4}}$, where n takes values in the

Scheme	Time (ns)
BCH3	68.9
EH3	1,798
RM7	$26.4 \cdot 10^6$

Table IV. Sketching time per interval.

range $[1 \dots 32]$.

The results in Figure 4 are not very encouraging. Only when the number of variables is greater than 25, the approximate algorithm needs less polynomial evaluations than the direct exhaustive method. But this implies intervals of extremely large size that are unlikely to appear in practice. If we also take into account the fact that the provided result is not exact and its error could propagate exponentially, we think that the approximate XOR-AND counting algorithm is not applicable to the range-summing problem.

4.6 Empirical Evaluation

We implemented the fast range-summable algorithms for the BCH3, EH3, and RM7 schemes and we empirically evaluated them with the same experimental setup as in Section 3.9. The evaluation procedure is based on 100 experiments that use a number of randomly generated intervals and an equal number of sketches chosen for each method such that the overall running time is in the order of minutes in order to obtain stable estimates of the running time per sketch. The results, depicted in Table IV, are the average of the 100 runs and have errors of at most 5%. Notice that the execution time of BCH3 for ranges is merely 7 times larger than the execution time for a single sketch (refer to Table II for the running times of individual sketches). This happens because, as we mentioned earlier, our algorithm for BCH3 is essentially $O(1)$. The Extended Hamming scheme EH3 has an encouraging running time of approximately 1.8 μs, thus about 550,000 such computations can be performed per second on a modern processor. The RM7 fast range-summable algorithm is completely impractical since only about 40 computations can be performed per second. This is due to the fact that the algorithm is quite involved (a significant number of systems of linear equations have to be formed and solved). Even if special techniques are used to reduce the running time, at most a 32 factor reduction is possible and the scheme would be still impractical.

The net effect of these experimental results and of the theoretical discussions in this section is that there is no practical fast range-summable algorithm for any of the known 4-wise generating schemes, while the algorithm for BCH3 is extremely efficient.

5. SIZE OF JOIN USING AMS-SKETCHES

As we pointed out earlier, an alternative solution to the size of join problem when at least one of the input relations is given as a set of intervals is to use fast range-summable families of random variables to speed-up the sketching of the intervals. Using the current understanding of the AMS-sketches applied to the size of join problem, the 4-wise independence of the random variables is required in order to keep the variance small (we show why latter in the section). Since, as it is shown in

Section 4, there is no practical fast range-summable 4-wise independent generating scheme, it looks like DMAP is the only feasible alternative, given the state-of-the-art. What we show in this section is that the Extended Hamming EH3 scheme [Feigenbaum et al. 2002] not only can be used as a reasonable replacement of the 4-wise independent schemes (this is the theoretical result proved in [Feigenbaum et al. 2002] for the L^1-difference of two streaming functions), but it is usually as good, in absolute big-O notation terms, and in certain situations significantly surpasses the 4-wise independent schemes for AMS-sketches solutions to the size of join problem. We provide here the theoretical support for this statement, and defer the empirical evidence to Section 6.

We proceed by taking a close look at the estimation of the size of join of two relations using AMS-sketches. As already presented in Section 2.1, the size of join $|R \bowtie_A S|$ of two relations R and S with a common attribute A is defined as $|R \bowtie_A S| = \sum_{i \in I} r_i s_i$. To estimate this quantity, we use a ± 1 family of random variables $\{\xi_i | i \in I\}$ that are at least 2-wise independent (but not necessarily more). We define the sketches, one for each relation, as $X_R = \sum_{i \in I} r_i \xi_i$ and $X_S = \sum_{i \in I} s_i \xi_i$. The random variable $X = X_R X_S$ estimates, on expectation, the size of join $|R \bowtie_A S|$. It is easy to show that, due to the 2-wise independence, $E[X] = \sum_{i \in I} r_i s_i$, which is exactly the size of join. Since all the generating schemes published in the literature (see Section 3 for a review of the most important ones) are 2-wise independent, they all produce estimates that are correct on average. The main difference between the schemes is the variance of X, $\mathrm{Var}(X)$. The smaller the variance, the better the estimation; in particular, the error of the estimate is proportional with $\frac{\sqrt{\mathrm{Var}(X)}}{E[X]}$.

We analyze now the variance of X. Following the technique in [Alon et al. 1996], we first compute $E\left[X^2\right]$ since $\mathrm{Var}(X) = E\left[X^2\right] - E[X]^2$. We have:

$$E\left[X^2\right] = E\left[\left(\sum_{i \in I} r_i \xi_i \sum_{i \in I} s_i \xi_i\right)^2\right] = \sum_{i \in I} \sum_{j \in I} \sum_{k \in I} \sum_{l \in I} r_i r_j s_k s_l E\left[\xi_i \xi_j \xi_k \xi_l\right]$$

The expression $E[\xi_i \xi_j \xi_k \xi_l]$ is equal to 1 if groups of two variables are the same (i.e., $i = j \wedge k = l$ or $i = k \wedge j = l$ or $i = l \wedge j = k$) since $\xi_i^2 = 1$ irrespective of the actual value of ξ_i ($1^2 = 1$ and $(-1)^2 = 1$). $E[\xi_i \xi_j \xi_k \xi_l]$ is equal to 0 if two variables are identical, say $i = j$, and two are different, since then $E[\xi_i \xi_j \xi_k \xi_l] = E[1 \cdot \xi_k \xi_l] = 0$, using the 2-wise independence property. The same is true when three of the variables are equal, say $i = j = k$, but the fourth one is not, since $E[\xi_i \xi_j \xi_k \xi_l] = E\left[\xi_i^3 \xi_l\right] = 0$ (we use the fact that $\xi_i^3 = \xi_i$). The above observations are not dependent on what generating scheme is used, as long as it is at least 2-wise independent. The contribution of the 1 values to $E\left[X^2\right]$ adds up to:

$$\sum_{i \in I} \sum_{j \in I} r_i^2 s_j^2 + 2 \cdot \left(\sum_{i \in I} r_i s_i\right)^2 - 2 \cdot \sum_{i \in I} r_i^2 s_i^2$$

In the expression of the variance $\mathrm{Var}(X)$, the middle term has the coefficient 1, instead of 2, since $E[X]^2$ is subtracted. The difference between the various generating schemes consists in what other terms have to be added to the above formula. The additional terms correspond only to the values of i, j, k, l that are all different

(otherwise the contribution is 1 or 0 irrespective to the scheme, as explained before). We explore what are the additional terms for three of the schemes, namely BCH5, BCH3, and EH3.

5.1 Variance for BCH5

The BCH5 scheme is 4-wise independent, which means that for $i \neq j \neq k \neq l$, we have:

$$E\left[\xi_i\xi_j\xi_k\xi_l\right] = E\left[\xi_i\right] \cdot E\left[\xi_j\right] \cdot E\left[\xi_k\right] \cdot E\left[\xi_l\right] = 0$$

since all the ξs are independent and $E\left[\xi_i\right] = 0$ for all generators. This means that no other terms are added to the above variance. The following formula results:

$$\mathrm{Var}\left(X\right)_{\mathrm{BCH5}} = \sum_{i\in I}\sum_{j\in I} r_i^2 s_j^2 + \left(\sum_{i\in I} r_i s_i\right)^2 - 2\cdot\sum_{i\in I} r_i^2 s_i^2 \tag{11}$$

5.2 Variance for BCH3

BCH3 is only 3-wise independent, thus clearly it is not the case that, for all $i \neq j \neq k \neq l$, $E\left[\xi_i\xi_j\xi_k\xi_l\right] = 0$. To characterize the value of the expectation for different variables, Proposition 4.3 is extensively applied.

LEMMA 5.1. *Assume the ξs are generated using the BCH3 scheme. Then, for $i \neq j \neq k \neq l$, $E\left[\xi_i\xi_j\xi_k\xi_l\right] = 0$ if $i \oplus j \oplus k \oplus l \neq 0$, and $E\left[\xi_i\xi_j\xi_k\xi_l\right] = 1$ if $i \oplus j \oplus k \oplus l = 0$, where $\oplus$ is the bitwise XOR.*

PROOF. Let $S = [s_0, S_1]$ be the $(n+1)$-bits random seed with s_0 its first bit and S_1 the last n bits. Using the notations and the definition of BCH3 in Section 3, we have:

$$\xi_i = (-1)^{s_0 \oplus S_1 \cdot i}$$

With this, we obtain:

$$\begin{aligned} E\left[\xi_i\xi_j\xi_k\xi_l\right] &= E\left[(-1)^{s_0\oplus S_1\cdot i}\cdot(-1)^{s_0\oplus S_1\cdot j}\cdot(-1)^{s_0\oplus S_1\cdot k}\cdot(-1)^{s_0\oplus S_1\cdot l}\right] \\ &= E\left[(-1)^{S_1\cdot i\oplus S_1\cdot j\oplus S_1\cdot k\oplus S_1\cdot l}\right] \\ &= E\left[(-1)^{S_1\cdot(i\oplus j\oplus k\oplus l)}\right] \end{aligned}$$

Using the result in Proposition 4.3 with $C = 0$ and $S_1, \ldots, S_n$ set as the last n bits of the seed S, we know that the expression $S_1 \cdot (i \oplus j \oplus k \oplus l)$ takes the values 0 and 1 equally often for random seeds S_1 when $i \oplus j \oplus k \oplus l \neq 0$ – this immediately implies that $E\left[\xi_i\xi_j\xi_k\xi_l\right] = 0$. When $i \oplus j \oplus k \oplus l = 0$, $S_1 \cdot (i \oplus j \oplus k \oplus l) = 0$ irrespective of S_1, giving $E\left[\xi_i\xi_j\xi_k\xi_l\right] = 1$. □

This implies that the variance for BCH3 contains an additional term besides the ones that appear in the variance formula for BCH5. The extra term has the following form:

$$\Delta\mathrm{Var}\left(\mathrm{BCH3}\right) = \sum_{i\in I}\sum_{j\in I, j\neq i}\sum_{k\in I, k\neq i,j} r_i r_j s_k s_{i\oplus j\oplus k}$$

since $i \oplus j \oplus k \oplus l = 0$ implies $l = i \oplus j \oplus k$. The additional term in the BCH3 variance can be significantly large, implying a big increase over the variance of BCH5. However, as we will see in the experimental section, the influence of the extra-term almost vanishes for high-skew data and the variance of BCH3 becomes comparable with the variance of BCH5.

EXAMPLE 5.2. *In order to give a clear image on the values $\Delta Var(BCH3)$ can take, we provide two extreme examples. First, consider that both relations R and S have uniform distributions with the value r, respectively s. This gives:*

$$\Delta Var(BCH3)_{uniform} = r^2 s^2 \sum_{i \in I} \sum_{j \in I, j \neq i} \sum_{k \in I, k \neq i,j} 1 \lessapprox r^2 s^2 |I|^3$$

value that is an order of $|I|$ greater than the BCH5 variance, thus dominating it and increasing the BCH3 variance to possibly extreme high values. Second, consider that both R and S are very skewed. Actually, there exists only one positive frequency in R (r), respectively S (s). The extra-variance becomes:

$$\Delta Var(BCH3)_{skewed} = rs \leq r^2 s^2$$

which is smaller than the corresponding BCH5 variance, thus it can be ignored.

5.3 Variance for EH3

As BCH3, the Extended Hamming scheme (EH3) is also 3-wise independent, which might suggest that the variance of EH3 is similar to the variance of BCH3 (extra terms not in the variance of BCH5 have to appear, otherwise the scheme would be 4-wise independent). As we show next, even though only positive terms appeared in the variance for BCH3, in the EH3 variance negative terms appear as well. These negative terms, in certain circumstances, can compensate completely for the positive terms and give a variance that becomes **zero**.

LEMMA 5.3. *Assume the ξs are generated using the EH3 scheme. Then, for $i \neq j \neq k \neq l$,*

$$E[\xi_i \xi_j \xi_k \xi_l] = \begin{cases} 0, & \text{if } i \oplus j \oplus k \oplus l \neq 0 \\ 1, & \text{if } i \oplus j \oplus k \oplus l = 0 \wedge \\ & \quad h(i) \oplus h(j) \oplus h(k) \oplus h(l) = 0 \\ -1, & \text{if } i \oplus j \oplus k \oplus l = 0 \wedge \\ & \quad h(i) \oplus h(j) \oplus h(k) \oplus h(l) = 1 \end{cases}$$

where $\oplus$ is the bitwise XOR.

PROOF. We know that

$$\xi_i = (-1)^{s_0 \oplus S_1 \cdot i \oplus h(i)}$$

for random variables generated using the EH3 scheme. Replacing this form into the expression for $E[\xi_i \xi_j \xi_k \xi_l]$ and applying the same manipulations as in the proof of Proposition 5.1, we get:

$$E[\xi_i \xi_j \xi_k \xi_l] = E\left[(-1)^{S_1 \cdot (i \oplus j \oplus k \oplus l) \oplus (h(i) \oplus h(j) \oplus h(k) \oplus h(l))}\right]$$

If we use again Proposition 4.3 with $C = h(i) \oplus h(j) \oplus h(k) \oplus h(l)$ and $S_1, \ldots, S_n$ set as the last n bits of the seed S, we first observe that the expectation is 0 when $i \oplus j \oplus k \oplus l \neq 0$. When $i \oplus j \oplus k \oplus l = 0$, the expected value is always $(-1)^C$. For BCH3 the constant C always took the value 0, thus the expectation in that case was always 1. For EH3, the value of C depends on the function h – it is 1 when $h(i) \oplus h(j) \oplus h(k) \oplus h(l) = 1$. This implies the value -1 for $E[\xi_i \xi_j \xi_k \xi_l]$. □

Using the above result, we observe that $E[\xi_i \xi_j \xi_k \xi_l] = -1$ when $i \oplus j \oplus k \oplus l = 0$ and $h(i) \oplus h(j) \oplus h(k) \oplus h(l) = 1$. This means that, even though EH3 can have all the 1 terms BCH3 has, it can also have -1 terms, thus, it can potentially compensate for the 1 terms. Indeed, the following results show that this is exactly what happens under certain independence assumptions.

In order to predict the performance of EH3, we perform an average-case analysis of the scheme. Since AMS-sketches are randomized schemes, the average-case analysis is a more appropriate characterization than a worst-case analysis. [Feigenbaum et al. 2002] provide a worst-case analysis of EH3 for the particular case of computing the L^1-difference of two functions. For the size of join problem, the situation is more complicated because the frequencies have to be considered too and the parameter we are interested in is the self-join size of each relation.

To obtain an average analysis, consider first the theorem:

THEOREM 5.4. *For i, j, k taking all the possible values over the domain $I = \{0, \ldots, 4^n - 1\}$, $n > 0$, the function $g(i, j, k) = h(i) \oplus h(j) \oplus h(k) \oplus h(i \oplus j \oplus k)$ takes the value 0 z_n times and the value 1 y_n times, where z_n and y_n are given by the following recursive equations:*

$$z_1 = 40, \quad y_1 = 24$$
$$z_n = 40 \cdot z_{n-1} + 24 \cdot y_{n-1}$$
$$y_n = 24 \cdot z_{n-1} + 40 \cdot y_{n-1}$$

PROOF. The base case, $n = 1$, can be easily verified by hand. The recursion is based on the observation that the groups of two bits from different h functions interact independently and give 40 zero values and 24 one values. When the results of two groups are XOR-ed, a zero is obtained if both groups are zero or both are one; a one is obtained if a group equals zero and the other group equals one. □

We have to characterize the behavior of function $g(i, j, k)$ for $i \neq j \neq k$ (when at least two of these variables are equal, we obtain the special case of the variance for BCH5 that we have already considered). The number of times at least two out of the three variables are equal is $eq_n = 3 \cdot (4^n)^2 - 2 \cdot 4^n$, which allows us to determine the desired quantities, i.e., the number of zeros is $z_n - eq_n$, while the number of ones is y_n. To determine the average behavior of EH3, we assume that neither the frequencies in R are correlated with the frequencies in S nor the frequencies in S are correlated amongst themselves. This allows us to model the quantity $l = i \oplus j \oplus k$ as a uniformly distributed random variable L over the domain I. Moreover, due to the same independence assumptions, function $g(i, j, k)$ can be modeled as a random variable G, independent of L, that has the same macro behavior as $g(i, j, k)$, i.e., it takes the values 0 and 1 the same number of times. With these random variables,

we get:

$$E\left[s_L\right] = \frac{1}{|I|}\sum_{l\in I} s_l$$

and

$$E\left[(-1)^G\right] = 1\cdot P[G=0] + (-1)\cdot P[G=1] = \frac{z_n - eq_n - y_n}{z_n - eq_n + y_n}$$

The expected value of the additional terms that appear in the variance of EH3 is given by:

$$\begin{aligned} E\left[\Delta\mathrm{Var}\left(\mathrm{EH3}\right)\right] &= E\left[\sum_{i\in I}\sum_{j\in I}\sum_{k\in I} r_i r_j s_k (-1)^G s_L\right] \\ &= \sum_{i\in I}\sum_{j\in I}\sum_{k\in I} r_i r_j s_k E\left[(-1)^G\right] E\left[s_L\right] \\ &= \frac{1}{|I|}\left(\sum_{i\in I} r_i\right)^2 \left(\sum_{i\in I} s_i\right)^2 \frac{z_n - eq_n - y_n}{z_n - eq_n + y_n} \end{aligned}$$

Overall, the variance of the EH3 generating scheme is:

$$\mathrm{Var}\left(X\right)_{\mathrm{EH3}} = \frac{1}{|I|}\left(\sum_{i\in I} r_i\right)^2 \left(\sum_{i\in I} s_i\right)^2 \frac{z_n - eq_n - y_n}{z_n - eq_n + y_n} + \mathrm{Var}\left(X\right)_{\mathrm{BCH5}} \qquad (12)$$

Notice that the additional term over the variance for BCH5 is inversely proportional with the size of the domain I. Also, the last factor in the additional term takes small sub-unitary values. The combined effect of these two is to drastically decrease the influence of the extra-term on the EH3 variance, making it close to the BCH5 variance. Actually, there exist situations for which the EH3 variance is significantly smaller than the BCH5 variance. It can even become equal to **zero**. The following corollary states this surprising result:

COROLLARY 5.5. *If $r_i = r$ and $s_i = s$, $i \in I$, with r and s constants, and $|I| = 4^n$, then:*

$$Var\left(X\right)_{EH3} = 0$$

The reason the variance of EH3 is **zero** when the distribution of both R and S is uniform is the fact that the -1 terms cancel out entirely the 1 terms. For less extreme cases, when the distribution of the two relations is close to a uniform distribution, EH3 significantly outperforms BCH5. This intuition is confirmed by the experimental results in Section 6.

Given the theoretical results in this section, the experimental results in the following section, and the fact that EH3 is fast range-summable and can be implemented more efficiently than BCH5, we recommend the exclusive use of the EH3 random variables for size of join estimations using AMS-sketches.

5.4 Fast Range-Summable Methods vs DMAP

Given the results characterizing the AMS-sketches variance, we provide a theoretical comparison between the fast range-summable methods and DMAP for estimating the size of join between an interval relation and a point relation. We consider the time to update the sketches and the accuracy as the parameters of the comparison.

There exists a logarithmic factor (in the size of the domain) in the update-time of the point relation between standard AMS-sketches, as is the case for fast range-summable schemes, and DMAP. This is the case because DMAP represents each point by all the dyadic intervals (of all ranges) that contain that point. The time to update the sketch of the interval relation is logarithmic in the size of the interval both for fast range-summable methods as well as for DMAP. More precisely, BCH3 has constant update-time in the average case, while EH3 has logarithmic update-time. Overall, the methods have the following update-times: BCH3 (constant-constant), EH3 (constant-logarithmic), and DMAP (logarithmic-logarithmic).

As we have seen, DMAP applies a number of transformations in order to efficiently sketch intervals. Although the size of join is invariant to these transformations, the variance of the sketch estimator changes in the new domain. Here we identify what is the relation between the variance of the size of join estimators in the two domains – the original domain and the domain of dyadic intervals.

The variance of the size of join estimator is bounded by twice the product of the self-join sizes of the involved relations, in our case R, the interval relation, and S, the point relation. In order to find a relationship between these two variances, relations between the individual factors, i.e., $\sum_i r_i^2$ and $\sum_\delta r_\delta^2$, respectively, $\sum_i s_i^2$ and $\sum_\delta s_\delta^2$, should first be determined.

$$\mathrm{Var}\left(|R \bowtie S|\right) \leq 2\mathrm{SJ}(R)\mathrm{SJ}(S) = 2\sum_i r_i^2 \sum_i s_i^2$$

$$\mathrm{Var}\left(|R_{\mathrm{DMAP}} \bowtie S_{\mathrm{DMAP}}|\right) \leq 2\mathrm{SJ}(R_{\mathrm{DMAP}})\mathrm{SJ}(S_{\mathrm{DMAP}}) = 2\sum_\delta r_\delta^2 \sum_\delta s_\delta^2$$

The ratio $\frac{\sum_\delta s_\delta^2}{\sum_i s_i^2}$ takes the smallest value, i.e., $n+1$, where $|I| = N = 2^n$, when the distribution of S is very skewed (there exists only one positive frequency, all the others being zero). The largest value of this ratio is $2^{n+1}-1$, for S having a uniform distribution. Intuitively, in the first case, only the set of dyadic intervals corresponding to the point with the non-zero frequency appear in the dyadic domain. For the second case, all dyadic intervals in the dyadic domain appear frequently often. While the extreme values for the ratio $\frac{\sum_\delta s_\delta^2}{\sum_i s_i^2}$ are greater than one, the extreme values of the ratio $\frac{\sum_\delta r_\delta^2}{\sum_i r_i^2}$ are smaller or equal than one. The largest value of the ratio is 1, when all the intervals in R are of size 1 (points). The smallest value, i.e., $\frac{1}{2^n}$, appears when all the intervals are of domain I size.

Table V summarizes the extreme values for the ratio between the variance of the size of join estimator in the dyadic domain and in the original domain. The range of this ratio varies from $\frac{\log N}{N}$ to $2N$, where values larger than 1 imply that the DMAP method increases the estimation error. The case for which DMAP has better error guarantees is when the intervals have large size (close to the domain

Interval size / Point distribution	Skewed	Uniform
Large	$\frac{\log N}{N}$	2
Small	$\log N$	$2N$

Table V. Extreme values for $\frac{\mathrm{Var}(|R_{\mathrm{DMAP}} \bowtie S_{\mathrm{DMAP}}|)}{\mathrm{Var}(|R \bowtie S|)}$.

size) and the point relation has a very skewed distribution. For all the other cases, DMAP worsens the error (the extreme is a factor of $2N$).

Since no method always dominates the other, distribution information on the input is required in order to provide a comparison. The scenario when DMAP wins does not seem likely to appear in practice. The experimental results in the next section confirm the fact that EH3 is expected to work better in most situations (EH3 is significantly better in all experiments we performed).

6. EMPIRICAL EVALUATION

The purpose of the empirical evaluation is threefold. First, we want to validate the theoretical models in Section 5, especially the average behavior of EH3. Second, we want to compare the BCH3, EH3, and BCH5 schemes for estimations using AMS-sketches. Third, we want to compare EH3 with DMAP [Das et al. 2004] (see Section 2.4) on the two applications introduced in Section 2, namely, size of spatial joins and selectivity estimation for histograms construction. The comparison with BCH3 is omitted since its error is significantly higher when compared to EH3 or DMAP. We do not perform comparisons with the fast range-summable version of the Reed-Muller scheme (RM7) since its throughput is not higher than 40 sketch computations per second (EH3 is capable of performing more than $550,000$ sketch computations per second as shown in Section 4).

The main findings of our experimental study can be summarized as follows:

—**Validation of the theoretical model for the EH3 generating scheme.** Our study shows that the behavior of the EH3 generating scheme is well predicted by the theoretical model in Section 5.

—**BCH3 vs EH3 vs BCH5.** EH3 has approximately the same error, or, in the case of low-skew distributions, a significantly better error than the BCH5 scheme. This justifies our recommendation to use EH3 instead of BCH5. For high-skew data, BCH3 has approximately the same error as BCH5 and EH3, making it the perfect solution for sketching intervals when the data is skewed (BCH3 is constant-time fast range-summable).

—**EH3 vs DMAP.** Our study shows that both for real and synthetic data applications, EH3 significantly outperforms DMAP in terms of the error, sometimes by as much as a factor of 8, with the same memory usage and comparable running time.

We performed the experiments using the setup in Section 3. We give detailed descriptions of the datasets and the comparison methodology for each group of experiments.

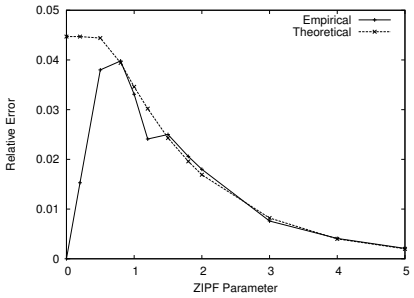

Fig. 5. EH3 error.

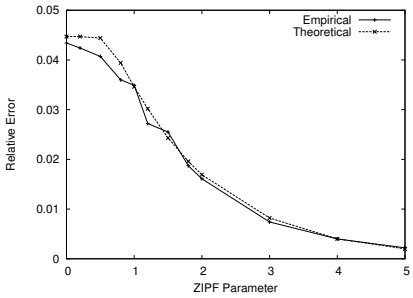

Fig. 6. BCH5 error.

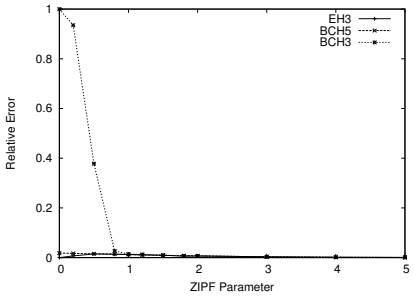

Fig. 7. Scheme comparison (full).

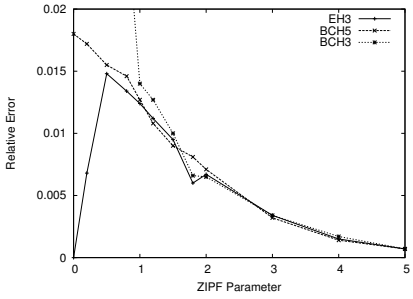

Fig. 8. Scheme comparison (detail).

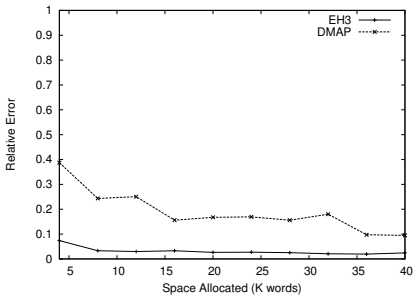

Fig. 9. LANDO ⋈ LANDC.

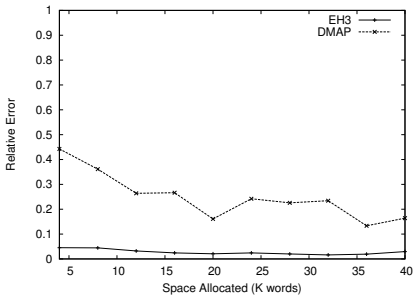

Fig. 10. LANDO ⋈ SOIL.

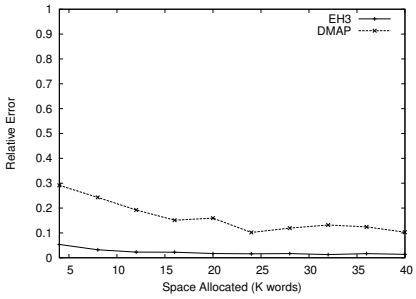

Fig. 11. LANDC ⋈ SOIL.

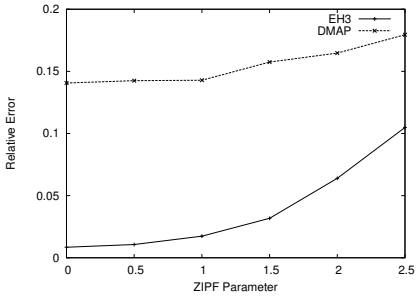

Fig. 12. Selectivity estimation.

6.1 Validation of the EH3 Model

To validate the average error formula in Equation (12), we generated Zipf distributed data with the Zipf coefficient ranging from 0 to 5 over domains of various sizes in order to estimate the self-join size. The prediction is performed using AMS-sketches with only one median, i.e., only averaging is used to decrease the error of the estimate. In Figure 5 we depict the comparison between the average error of the EH3 scheme and the theoretical prediction given by Equation (12) for a domain with the size $16,384$ and a relation containing $100,000$ tuples. Notice that, when the value of the Zipf coefficient is larger than 1, the prediction is accurate. When the Zipf coefficient is between 0 and 1, the error of EH3 is much smaller (it is zero for a uniform distribution). This is explained in Corollary 5.5, which states that the variance of EH3 is zero when the distribution of the data is uniform and the size of the domain is a power of 4. For completeness, we depict the comparison between the theoretical model and the experimental results for BCH5 scheme in Figure 6.

6.2 BCH3 vs EH3 vs BCH5

We performed the same experiments as in the previous section for BCH3, EH3, and BCH5, except that the number of medians was fixed to 10. The results of the experiments are depicted in Figure 7 (full-size picture) and Figure 8 (detailed picture focusing on EH3 and BCH5). Notice that the errors of BCH3, EH3, and BCH5 are virtually the same for Zipf coefficients greater than 1. While the EH3 error is significantly smaller for Zipf coefficients lower than 1, the BCH3 error can get extremely high values (100% relative error). These results confirm our theoretical characterizations both for BCH3 and EH3 schemes.

When compared to the results in Figure 5, the errors are smaller by a factor of 3. This is due to the fact that 10 medians were used instead of only 1 and the medians have almost the same effect in reducing the error as the averages – the same observation was made in [Das et al. 2004].

6.3 EH3 vs DMAP for Spatial Joins

We used the same experimental setup as in [Das et al. 2004] to compare EH3 and DMAP for approximating the size of spatial joins. Three datasets are used: LANDO, describing land cover ownership for the state of Wyoming and containing $33,860$ objects; LANDC, describing land cover information such as vegetation types for the state of Wyoming and containing $14,731$ objects; and SOIL, representing the Wyoming state soils at a $1 : 10^5$ scale and containing $29,662$ objects. The average error for estimating the size of spatial joins for both EH3 and DMAP is depicted in Figure 9, 10, and 11. The sketch size varies between 4 and 40 K words of memory. Notice that in all the experiments EH3 significantly outperforms DMAP by as much as a factor of 8. This means that DMAP needs as much as 64 times more memory in order to achieve the same error guarantees.

Table VI contains the timing results for sketching intervals using the two practical fast range-summable schemes (BCH3 and EH3) and the DMAP method. We present the average time for sketching an interval from each of the real datasets. As expected, BCH3 is the fastest scheme both for sketching only the interval, but

Scheme	Interval			Interval + End-Points		
	LANDC	LANDO	SOIL	LANDC	LANDO	SOIL
BCH3	50	75	79	106	126	115
EH3	637	651	604	681	701	651
DMAP	409	298	309	1500	1401	1391

Table VI. Sketching time per interval (ns).

also for sketching the interval as well as its end-points. The time to sketch an interval using DMAP is about half the time used by the EH3 fast range-summable algorithm. When sketching both the interval and its end-points, as is the case for the size of join between an interval relation and a point relation, the ratio reverses – DMAP uses twice as much time as EH3. This happens because DMAP uses a logarithmic number of points (in the size of the domain) to represent each interval end-point, while EH3 has to generate only two random variables, one for each end-point.

The difference between the results reported in Table VI and the previous results (Table II and IV) is due to the timing procedure. While for the previous results we measured only the generating/fast range-summing time, the results in Table VI also include the overhead time (routine invocation, etc.).

6.4 EH3 vs DMAP for Selectivity Estimation

To compare EH3 and DMAP on the task of selectivity estimation, we used the synthetic data generator from [Dobra et al. 2002]. It generates multi-dimensional data distributions consisting in regions, randomly placed in the two-dimensional space, with the number of points in each region Zipf distributed and the distribution within each region Zipf distributed as well. For the experiments we report here, we generated two-dimensional datasets with the domain for each dimension having the size 1024. A dataset consists of 10 regions of points. The distribution of the frequencies within each region has a variable Zipf coefficient, as shown in Figure 12. Notice that for small Zipf coefficients EH3 outperforms DMAP by a factor of 14. When the Zipf coefficient becomes larger, the gap between DMAP and EH3 shrinks considerably, but EH3 still outperforms DMAP by a large margin.

7. CONCLUSIONS

In this paper we conducted both a theoretical as well as an empirical study of the various schemes used for the generation of the random variables that appear in AMS-sketches estimations. Our primary focus was the identification of the fast range-summable schemes that can sketch intervals in sub-linear time. We explain how the fast range-summable versions of two of the 3-wise independent schemes, BCH3 and EH3, can be implemented efficiently and we provide an empirical comparison with the only known 4-wise independent fast range-summable scheme (RM7) that reveals that only BCH3 and EH3 are practical. Moreover, we provide theoretical and empirical evidence that EH3 can replace the 4-wise independent schemes for the estimation of the size of join using AMS-sketches. The EH3-based solutions significantly outperform the state-of-the-art DMAP algorithms for applications such as the size of spatial joins and the dynamic construction of

histograms.

The main recommendation of this paper is to use the EH3 random variables for AMS-sketches estimations of the size of join since they can be generated more efficiently and use smaller seeds than any of the 4-wise independent schemes. At the same time, the error of the estimate is as good as or, in the case when the distribution has low skew, better than the error provided by a 4-wise independent scheme. In the same time, BCH3 is the perfect solution for sketching highly-skewed interval data since we provide a constant-time algorithm for range-summing BCH3 random variables.

REFERENCES

Aduri, P. and Tirthapura, S. 2005. Range efficient computation of F_0 over massive data streams. In *Proceedings of the twenty-first IEEE ICDE International Conference on Data Engineering*. IEEE Computer Society, Washington, DC, USA, 32–43.

Alon, N., Babai, L., and Itai, A. 1986. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms 7,* 4, 567–583.

Alon, N., Gibbons, P. B., Matias, Y., and Szegedy, M. 2002. Tracking join and self-join sizes in limited storage. *Journal of Computer and System Sciences 64,* 3, 719–747.

Alon, N., Matias, Y., and Szegedy, M. 1996. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM Symposium on Theory of Computing*. ACM Press, New York, NY, USA, 20–29.

Bar-Yosseff, Z., Kumar, R., and Sivakumar, D. 2002. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the thirteenth annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 623–632.

Calderbank, A. R., Gilbert, A., Levchenko, K., Muthukrishnan, S., and Strauss, M. 2005. Improved range-summable random variable construction algorithms. In *Proceedings of the sixteenth annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 840–849.

Carter, L. and Wegman, M. N. 1979. Universal classes of hash functions. *Journal of Computer and System Sciences 18,* 2, 143–154.

Charikar, M., Chen, K., and Farach-Colton, M. 2004. Finding frequent items in data streams. *Theoretical Computer Science 312,* 1, 3–15.

Das, A., Gehrke, J., and Riedewald, M. 2004. Approximation techniques for spatial data. In *Proceedings of the twenty-third ACM SIGMOD International Conference on Management of Data*. ACM Press, New York, NY, USA, 695–706.

Deskins, W. E. 1996. *Abstract Algebra*. Dover Publications, New York, NY, USA.

Dobra, A., Garofalakis, M., Gehrke, J., and Rastogi, R. 2002. Processing complex aggregate queries over data streams. In *Proceedings of the twenty-first ACM SIGMOD International Conference on Management of Data*. ACM Press, New York, NY, USA, 61–72.

Ehrenfeucht, A. and Karpinski, M. 1990. The computational complexity of (*xor-and*) counting problems. Tech. Rep. TR-90-033, ICSI, Berkeley, CA, USA.

Feigenbaum, J., Kannan, S., Strauss, M., and Viswanathan, M. 2002. An approximate L^1-difference algorithm for massive data streams. *SIAM Journal on Computing 32,* 1, 131–151.

Ganguly, S., Garofalakis, M., and Rastogi, R. 2003. Processing set expressions over continuous update streams. In *Proceedings of the twenty-second ACM SIGMOD International Conference on Management of Data*. ACM Press, New York, NY, USA, 265–276.

Ganguly, S., Garofalakis, M., and Rastogi, R. 2004. Processing data-stream join aggregates using skimmed sketches. In *Proceedings of the nineth International Conference on Extending Database Technology*. Springer, 569–586.

Gilbert, A. C., Kotidis, Y., Muthukrishnan, S., and Strauss, M. J. 2003. One-pass wavelet decompositions of data streams. *IEEE Transactions on Knowledge and Data Engineering 15,* 3, 541–554.

Gilbert, A. C., Kotidis, Y., Muthukrishnan, S., and Strauss, M. J. 2005. Domain-driven data synopses for dynamic quantiles. *IEEE Transactions on Knowledge and Data Engineering 17,* 7, 927–938.

Goldreich, O. 1997. A sample of samplers - a computational perspective on sampling. *Electronic Colloquium on Computational Complexity (ECCC) 4,* 20.

Hedayat, A. S., Sloane, N. J. A., and Stufken, J. 1999. *Orthogonal Arrays: Theory and Applications*. Springer-Verlag, New York, NY, USA.

Karpinski, M. and Luby, M. 1993. Approximating the number of zeroes of a GF[2] polynomial. *Journal of Algorithms 14,* 2, 280–287.

Kempe, D., Dobra, A., and Gehrke, J. 2003. Gossip-based computation of aggregate information. In *Proceedings of the fourty-fourth Annual IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society, Washington, DC, USA, 482–491.

Krishnamurthy, B., Sen, S., Zhang, Y., and Chen, Y. 2003. Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the third ACM SIGCOMM Conference on Internet Measurement*. ACM Press, New York, NY, USA, 234–247.

Levchenko, K. 2005. `http://www.cse.ucsd.edu/~klevchen/II-2005.pdf`.

Luby, M. and Wigderson, A. 1995. Pairwise independence and derandomization. Tech. Rep. UCB/CSD-95-880, EECS UC Berkeley, Berkeley, CA, USA.

Shao, J. 1999. *Mathematical Statistics*. Springer-Verlag, New York, NY, USA.

Thaper, N., Guha, S., Indyk, P., and Koudas, N. 2002. Dynamic multidimensional histograms. In *Proceedings of the twenty-first ACM SIGMOD International Conference on Management of Data*. ACM Press, New York, NY, USA, 428–439.

Thorup, M. and Zhang, Y. 2004. Tabulation based 4-universal hashing with applications to second moment estimation. In *Proceedings of the fifteenth annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 615–624.

Wegman, M. and Carter, J. 1981. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences 3,* 22, 265–279.