AofA'12 DMTCS proc. **AQ**, 2012, 323–338

Data Streams as Random Permutations: the Distinct Element Problem

Ahmed Helmi[1], Jérémie Lumbroso[2,3], Conrado Martínez[1] and Alfredo Viola[4]

[1] *Departament Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, E-08034 Barcelona, Spain.*
[2] *Équipe APR, Laboratoire d'Informatique de Paris 6, Université Pierre et Marie Curie, F-75005 Paris, France.*
[3] *Projet Algorithms, INRIA Rocquencourt, F-78153 Les Chesnay, France.*
[4] *Instituto de Computación, Universidad de la República, Montevideo, Uruguay.*

In this paper, we show that data streams can sometimes usefully be studied as random permutations. This simple observation allows a wealth of classical and recent results from combinatorics to be recycled, with minimal effort, as estimators for various statistics over data streams.

We illustrate this by introducing RECORDINALITY, an algorithm which estimates the number of distinct elements in a stream by counting the number of k-records occurring in it. The algorithm has a score of interesting properties, such as providing a random sample of the set underlying the stream. To the best of our knowledge, a modified version of RECORDINALITY is the first cardinality estimation algorithm which, in the random-order model, uses neither sampling nor hashing.

Keywords: data stream, probabilistic algorithm, random permutation, random-order model.

Given a large *multiset* $S = (x_1, x_2, \ldots)$ of size $N = |S|$ (the total number of elements in S) which may contain repetitions, we are interested in extracting a basic quantity, the number n of *distinct* elements contained in S—which may alternatively be referred to as the *cardinality*—using linear time and sublinear memory, typically $O(\log n)$ or $O(\log \log n)$.

Why estimate the number of distinct elements? In the early 1980s, the database community led by IBM's researchers, seeking to abstract the low-level details of database querying, devised optimization schemes that required the number of distinct elements in a table or returned by an intermediate query. For instance, the number of distinct elements in a query's results helps to decide how to organize several chained queries—much like the better known problem of deciding in which order to multiply matrices.

But using the *exact* count is not desirable for at least two reasons: first of all, these queries are usually just one out of potentially many (nowadays, a single page view on Internet can sometimes involve hundreds of queries on the server's database), so systematically calculating an exact count is too *costly*; second, since the count is often ultimately used to make a binary decision—should I use method A or method B?—or at the very least a decision involving only a handful of choices, being off by 10% is generally not going to affect the outcome, so calculating an exact count is not even *useful*.

As a consequence, efficiently estimating the number of distinct elements quickly became a useful and relevant problem, first solved by Flajolet and Martin [13] in 1985. Eventually, it became obvious that estimating the number of distinct elements could also be useful in a wide range of fields, from data-mining to network security.

Prior work. This simple problem, called the *distinct element problem* or *cardinality estimation problem*, encapsulates many of the challenges of data streaming algorithms[(i)]—and is indeed the first problem which was framed in terms of streaming algorithms. Since Flajolet and Martin's algorithm *Probabilistic Counting*, many theoretical results have been published, and many algorithms have been designed: some incrementally improving effiency, others introducing new points of view.

Some algorithms make statistical hypotheses on the data stream; others first sample values from the stream and then make an estimate based on this sample (generally assuming that the nature of the repetitions in the sample can be scaled up to the entire stream); others still use *hash functions*—either as a discrete source (hash values as binary words of 0s and 1s) or as a real-valued source.

From a practical standpoint, the best algorithm that has been obtained is HYPERLOGLOG [12]. It is very fast, simple to implement, and reaches an accuracy of $1.03/\sqrt{m}$ using m small words of 5 bytes to estimate cardinalities up to 2^{40}. It is based on hash functions[(ii)].

Data streams as random permutations. In this paper, we introduce a new way of considering data streams: under a certain setting (which we detail at the beginning of Section 2), data streams can be studied as though they are *random permutations*.

First, this allows us to borrow from the vast literature on random permutations, and easily adapt existing results into new estimators. For instance, RECORDINALITY bases itself on the statistic of k-records; and since a limit distribution is known for this statistic, we rather effortlessly have a limit distribution for our algorithm as well. This is interesting, because prior algorithms have typically been too complicated to fully analyze in distribution, and Lumbroso [21] gave one of the few limit distributions for such algorithms.

Second, many statistics on random permutation only take into account the *relative ranks* of elements; this property translates to algorithms based on these statistics, which therefore only take into account the relative ranks of the hashed elements. This contrasts with existing cardinality estimation algorithms based on hash functions, which actually require the *value* of the hash function. We argue in Subsection 4.3 that this allows us to consider dispensing with hash functions altogether, and providing the first practical distinct element estimation algorithm in the random-order model. This model has gained popularity ever since [22] and while the distinct element problem has already been considered in this setting, to the best of our knowledge, only theoretical lower-bounds exist [5, 28].

Finally, this approach seems to hold a few interesting prospects. One of them is that of *sampling*; in this paper, we show that RECORDINALITY can be used as a *distinct sampling* algorithm in line with [9, 15, 20], and briefly introduce another similar algorithm in Subsection 4.1, which is based on another statistic, that presents the advantage of producing samples of size that is not constant, but logarithmic in n the number of distinct elements. This topic is something we wish to expand on in future versions of this work.

As far as we know, considering data streams as random permutations is a novel idea: in the past, this has only been touched upon in papers considering the *sortedness* or *monotonicity* of a data stream, expressed as the number of inversions; see for instance Ajtai *et al.* [1], Gopalan *et al.* [16], or Ergun and Jowhari [8].

(i) For a recent point of view on data streaming algorithms, one could not recommend Muthukrishnan's monography enough [26]!

(ii) For additional references, we recommend the bibliography of a nice informal survey by Flajolet [11].

Organisation. The main subject of study is an algorithm which we have named RECORDINALITY. Section 1 introduces the statistic on random permutations that this algorithm is based on, and recalls the existing results we make use of. We then, in Section 2, detail why data streams can be seen as random permutations, and state the main theorems on RECORDINALITY with their proofs—thus showing unbiasedness, providing accuracy, and also the limit distribution. Section 3 briefly presents some simulations, both to validate the theoretical results and to compare RECORDINALITY to similar existing algorithms. Finally, Section 4 expands on a few interesting properties both of our algorithm and our approach, such as their application to random sampling, or how they can be extended to the random-order model.

1 Records in permutations

The main algorithm we describe in this paper is based on a common generalization of the notion of *record*, or *left-to-right maximum* (which is a special case of this generalization, for $k = 1$).

Definition 1. Let σ be a permutation. The rank i is a k-record of the permutation, $k \in \mathbb{Z}_{>}$, iff

$$\#\left\{j \ : \ j < i \text{ and } \sigma_j > \sigma_i\right\} < k.$$

This definition may also be seen from an algorithmic perspective: while reading a permutation (in one-line notation) sequentially from left to right, we keep a table containing, at any given moment, the k largest elements which have been seen; in this setting, an element is a k-record if it has at some point been in the table (regardless of whether or not it is later removed). The number of k-records is the number of updates—insertion or replacement—made to the table.

Proposition 1 (Archibald and Martínez)**.** *Let r_k denote the number of k-records in a uniformly random permutation of size n. As n tends to infinity, with $k < n$, the expected value of r_k is*

$$\mathbb{E}_n[r_k] = k(H_n - H_k + 1) = k(\log n - \log k + 1) + O(1)\,.$$

Proposition 2 (Helmi, Martínez and Panholzer)**.** *Let r_k denote the number of k-records in a permutation of size n. The exact distribution of r_k is*

$$\mathbb{P}_n[r_k = j] = \begin{cases} [\![n = j]\!] & \text{if } k > n, \\ k^{j-k}\dfrac{k!}{n!}\begin{bmatrix} n-k+1 \\ j-k+1 \end{bmatrix} & \text{if } k \leqslant j \leqslant n \end{cases}$$

where we use the standard bracket notation for signless Stirling numbers of the first kind (see [17] for details on this notation).

There is a vast literature dealing with records on permutations, but our initial interest in this subject came from the *hiring problem* recently introduced by Broder *et al.* [4] (itself a generalization of the earlier, well-known *secretary problem* stated by Freeman [14] in 1983): we sequentially consider candidates rated with some score; for each candidate, we must decide whether to hire him or not, based solely on his score and the scores of previously seen candidates, and this decision is final; the goal is to hire the best possible staff. The results we reuse here, from papers by Archibald and Martínez [2], and by Helmi *et al.* [18], originally had to do with the "elite" strategy, under which candidates are only hired if they are locally optimal—and under this framework, the number of k-records is the size of the hired staff.

2 Recordinality: using records to estimate cardinality

The statistic we defined in the previous section is *insensitive to repetitions*: indeed the strict inequality in the definition of k-records ensures that in the presence of *repetitions*, only the first occurrence of a value can possibly be considered a k-record. This argument is crucial, because it means that when we sequentially read a stream to find k-records (or any other statistic which is similarly insensitive to repetitions[(iii)]), we can consider that the stream is a *permutation* of the underlying set of its distinct elements.

But we also use a random *hash function*, which we consider to be a function h which takes an element of the stream and returns some independent uniform $(0, 1)$ random variable[(iv)] associated with that element. Instead of reading elements of the stream, we read—and compare—hashes of these elements, thus uniform random variables. Because of this, we can consider that the permutation provided by the stream is a *random permutation*.

From this point on, we identify the stream with a random permutation, and we consider that r_k is the empirically measured number of records in that permutation, distributed as stated in Propositions 1 to 2.

Description of the algorithm. In RECORDINALITY, we scan the data stream and use a data structure to maintain the k largest values, while keeping track of the number of times a new element is inserted in the data structure. As we have said before, this number of changes, which we note r_k, is the number of k-records. We then estimate the number of distinct elements by applying the following estimator.

Theorem 1. *Let $r_k \in \mathbb{N}$ be the empirically observed number of k-records; the estimator $\mathcal{R}_k$ defined by*

$$\mathcal{R}_k := k \cdot \left(1 + \frac{1}{k}\right)^{r_k - k + 1} - 1$$

is an unbiased *estimator of the number n of distinct elements.*

Theorem 2. *The accuracy of estimator $\mathcal{R}_k$, expressed in terms of standard error, asymptotically satisfies*

$$\mathbb{SE}_n[\mathcal{R}_k] \sim \sqrt{\left(\frac{n}{k\mathrm{e}}\right)^{\frac{1}{k}} - 1}.$$

Finally, the distributional result given in Subsection 2.2 allows us to provide this corollary which, while not rigorous, restates the properties of the distribution in more immediate terms.

Corollary 1. *Let σ be the standard deviation of the distribution for some large k; the algorithm* RECORDINALITY *is expected to provide estimates within σ, 2σ, 3σ of the exact count in respectively* at least *68%, 95% and 99% of all cases.*

When k is smaller, the estimates may be significantly more concentrated. For instance, for $k = 10$, the estimates are within σ, 2σ, 3σ of the exact count in respectively 91%, 96% and 99% of all cases.

(iii) One example of a statistic that *is* sensitive to repetitions is the number of descents: indeed, in this case, $(1, 2, 1, 2, 1, 2, \ldots)$ can have as many descents as there are repetitions of the sequence $1, 2$.

(iv) While from a theoretical perspective, hash functions are clearly not random, empirically, even simple ones have proven to give a sufficiently good approximation of randomness for the assumption that they provide uniform random variables to hold for our purposes; a recent paper by Mitzenmacher and Vadhan [23] brings some theoretical grounding as to why that is.

2.1 Analysis

Proof. Our starting point is the exact formula of the probability distribution of the number of k-records in a random permutation, as stated in Proposition 2,

$$\mathbb{P}_n[r_k = j] = \begin{cases} [\![n = j]\!] & \text{if } k > n, \\ k^{j-k} \dfrac{k!}{n!} \begin{bmatrix} n-k+1 \\ j-k+1 \end{bmatrix} & \text{if } k \leqslant j \leqslant n. \end{cases}$$

Since the expected value of r_k, given by Proposition 1, is

$$\mathbb{E}_n[r_k] = k(\log n - \log k + 1) + O(1)$$

we expect $\mathcal{Z}_k$ to provide a rough estimate of n the number of distinct elements,

$$\mathcal{Z}_k := \exp(\alpha_k \cdot r_k),$$

with α_k some corrective factor to be determined. If we assume $k < n$, the expected value of $\mathcal{Z}_k$ is then simply,

$$\mathbb{E}_n[\mathcal{Z}_k] = \sum_{i=k}^{n} \exp(\alpha_k \cdot i) \cdot \mathbb{P}_n[r_k = i] = \frac{k! \exp((k-1)\alpha_k)}{k \cdot n!} \sum_{j=1}^{n-k+1} \begin{bmatrix} n-k+1 \\ j \end{bmatrix} (k \exp(\alpha_k))^j . \quad (1)$$

Recall the following identity for the unsigned Stirling numbers of the first kind [17],

$$\sum_{j=1}^{N} \begin{bmatrix} N \\ j \end{bmatrix} z^j = z(z+1)\cdots(z+N-1) = z^{\overline{N}},$$

where $z^{\overline{N}}$ is a standard notation for the *rising factorial*. By applying this identity to (1), we get

$$\mathbb{E}_n[\mathcal{Z}_k] = \frac{k! \exp((k-1)\alpha_k)}{k \cdot n!} (k \exp(\alpha_k))^{\overline{n-k+1}} .$$

We need to choose α_k so as to cancel out the $n!$ in the denominator, and to make sure there is some linear factor of n remaining (otherwise the estimator would be useless). Taking α_k such that $k \exp(\alpha_k) = k+1$ in turn means that

$$(k \exp(\alpha_k))^{\overline{n-k+1}} = \frac{(n+1)!}{k!}.$$

Thus letting $\alpha_k = \log(1 + 1/k)$, we can now considerably simplify our calculations

$$\mathbb{E}_n[\mathcal{Z}_k] = (n+1) \cdot \frac{1}{k} \left(1 + \frac{1}{k}\right)^{k-1}$$

and define the following unbiased estimator,

$$\mathcal{R}_k := k \left(1 + \frac{1}{k}\right)^{1-k} \mathcal{Z}_k - 1 = k \left(1 + \frac{1}{k}\right)^{r_k - k + 1} - 1.$$

□

2.2 Limit distribution

Since the estimator $\mathcal{R}_k$ is based on some statistic which has normal distribution and to which an exponential function is applied, it is natural to expect the estimator to have a log-normal distribution.

Theorem 3. *For any $k \geq 1$, the estimator $\mathcal{R}_k$ satisfies*

$$\mathcal{R}_k + 1 - k\left(1+\frac{1}{k}\right)^{-k+1} \text{Log-}\mathcal{N}(\mu_k, \sigma_k) \xrightarrow{(d)} 0$$

where

$$\mu_k = \log\left(1+\frac{1}{k}\right)\exp\left(r_k\right) \qquad \textit{and} \qquad \sigma_k = \log\left(1+\frac{1}{k}\right)\sqrt{\mathbb{V}_n[r_k]}.$$

3 Experimental analysis

This brief section has two aims: first, to show that our theoretical results are validated by practical simulations; second, to give some idea of how RECORDINALITY compares against similar cardinality estimation algorithms.

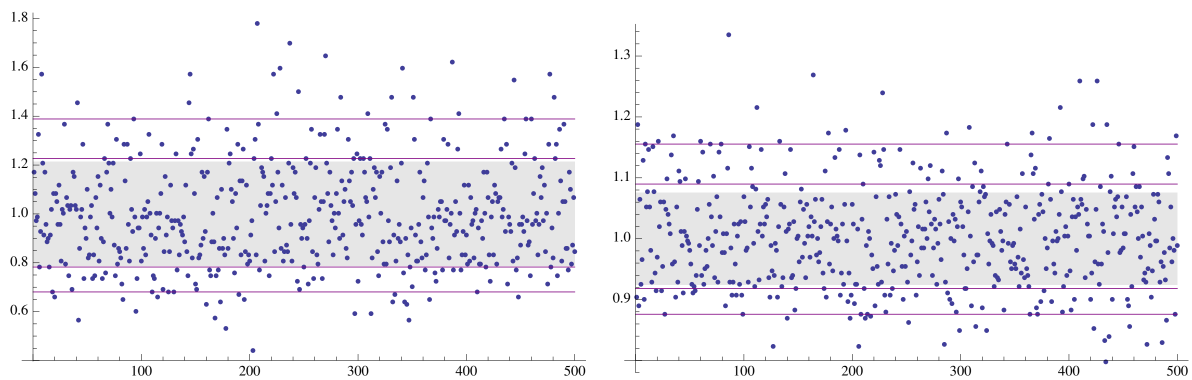

Fig. 1: Two plots showing the accuracy of 500 estimates of the number of distinct elements contained in Shakespeare's *A Midsummer Night's Dream*, on the left for $k = 64$ and on the right for $k = 256$. Above the top and below the bottom line lie 5% of the estimates; contained in between the two centermost lines is 70% of the estimates. As a reference, the gray rectangle delimits the area within one standard deviation from the mean.

Unbiasedness and standard error. In both plots of Figure 1, we have plotted the accuracy of 500 estimates of the number of distinct elements contained in Shakespeare's *A Midsummer Night's Dream*, each made with a new random hash function[(v)]; the accuracy is expressed on the y-axis as the ratio of the estimate to the actual number of distinct elements, which is $n = 3031$.

As stated by Theorem 1, RECORDINALITY is an unbiased estimator, which can be seen from the fact that the points on the plot are concentrated around 1.0. The gray rectangle delimits the area where estimates are within one standard deviation from the mean, as stated by Theorem 2; this mostly coincides with

[(v)] We use an affine transformation of a base hash function, using large prime parameters.

the area between the second and third (empirically placed) level lines containing 70% of the estimates, in accordance with Corollary 1.

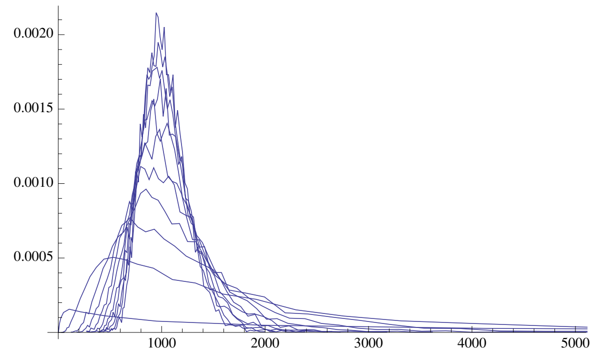

Fig. 2: Each curve is the empirical distribution of 10 000 estimates made by RECORDINALITY of some random text containing $n = 1000$ distinct elements, for $k = 1, 5, 10, \ldots, 50$. This validates theoretical calculations showing that estimates made by the algorithm are log-normally distributed.

Distribution. Figure 2 displays the empirical distribution made from 10 000 estimates on some random text using RECORDINALITY, for values of $k = 1, 5, 10, \ldots, 50$. It convincingly shows that the estimates are log-normally distributed.

Comparative performances. Tables 1, 2 and 3 compare the outcome of several thousand simulations using four different algorithms, on various types of data. For Shakespeare's play, where $n = 3031$, and where we required hashing, we would proceed as follow: for each simulation, we would draw a random hash function, apply it to the words of the stream, then feed the same hashed stream to each of the four algorithms. For the two random streams, containing respectively $n = 6000$ and $n = 50\,000$ distinct elements, we directly generated random uniform variables and used them as input for the four algorithms.

The algorithms we chose to compare RECORDINALITY to, were picked for the following reasons: *Adaptive Sampling* [9] is the only other cardinality estimation algorithm which, in addition, provides a random sample of the underlying set of the stream (see Subsection 4.1), so it seemed pertinent to compare its performance with that of our algorithm; the algorithm based on the k-th order statistic [3] functions maintaining the same data structure as RECORDINALITY and practically the same information; finally, HYPERLOGLOG [12] is the optimal algorithm used in practice.

4 Extensions

Up until now, we have presented RECORDINALITY (and our approach of data streams as random permutations) essentially following the criteria set by existing algorithms to estimate cardinality.

In this section, we expand the discussion to include some of the features which we believe to be more specific to our approach. In so doing, we give some good indication as to the ideas we wish to develop further.

k	RECORDINALITY		*Adaptive Sampling*		k-th Order Statistic		HYPERLOGLOG	
	Avg.	Error	Avg.	Error	Avg.	Error	Avg.	Error
4	2737	1.04	3047	0.70	4050	0.89	2926	0.61
8	2811	0.73	3014	0.41	3495	0.44	3147	0.42
16	3040	0.54	3012	0.31	3219	0.28	2981	0.26
32	3010	0.34	3078	0.20	3159	0.18	3001	0.18
64	3020	0.22	3020	0.15	3071	0.12	3011	0.13
128	3042	0.14	3032	0.11	3070	0.10	3031	0.09
256	3044	0.08	3027	0.07	3037	0.06	3025	0.06
512	3043	0.04	3043	0.05	3046	0.04	2975	0.08

Tab. 1: Estimating the number of distinct elements in Shakespeare's *A Midsummer Night's Dream* ($n = 3031$). The average and the empirical standard deviation divided by n, are calculated over 10 000 simulations.

k	RECORDINALITY		*Adaptive Sampling*		k-th Order Statistic		HYPERLOGLOG	
	Avg.	Error	Avg.	Error	Avg.	Error	Avg.	Error
4	5569	1.35	5826	0.67	7715	0.86	6148	0.70
8	6162	1.06	5899	0.42	6677	0.43	5938	0.40
16	6278	0.64	6008	0.31	6381	0.28	6131	0.31
32	6172	0.39	5930	0.21	6172	0.19	6058	0.19
64	6009	0.23	5974	0.15	6104	0.13	5949	0.13
128	5993	0.14	5974	0.10	6050	0.09	5996	0.09

Tab. 2: Similar experiments for a random stream containing $n = 6000$ distinct elements.

k	RECORDINALITY		*Adaptive Sampling*		k-th Order Statistic		HYPERLOGLOG	
	Avg.	Error	Avg.	Error	Avg.	Error	Avg.	Error
4	43658	1.19	59474	0.94	81724	1.30	44302	0.42
8	35230	0.52	47432	0.38	57028	0.41	52905	0.39
16	57723	0.98	49889	0.29	52990	0.23	51522	0.27
32	48686	0.45	49480	0.23	50556	0.18	48009	0.16
64	47617	0.34	50524	0.14	51146	0.13	49345	0.14
128	50097	0.17	50452	0.09	50947	0.08	51531	0.10
256	51742	0.11	50857	0.06	50348	0.06	49287	0.06
512	49496	0.09	49920	0.06	50084	0.04	49016	0.04

Tab. 3: Experiments for a random stream containg $n = 50\,000$ distinct elements—here 25 000 simulations were run.

4.1 Sampling algorithms

Adaptive Sampling is a very elegant algorithm, originally due to Mark Wegman, which was analyzed by Flajolet [9, 10], who discovered it could be used to yield interesting statistics beyond the number of distinct elements it was initially designed to estimate.

Indeed, at any point during the course of its execution, the algorithm (parameterized to use m words of memory) stores a uniform sample of between $m/2$ and m distinct elements from the set underlying the stream—that is to say elements are sampled independently of their frequency in the stream: an element appearing a thousand times, and another appearing only once would have the same probability of being in the sample. By attaching frequency counters to the elements, the proportion of various classes of elements can be estimated: for instance, those elements appearing once (called *mice*, in network analysis) or those appearing more than say, ten times (called *elephants*), see [19, 20] for detailed analyses.

This algorithm was subsequently rediscovered by several authors: by Bar-Yossef *et al.* in [3, §4], but also, and perhaps most famously, by Gibbons [15], who most pertinently renamed it *Distinct Sampling*. More recently, Monemizadeh and Woodruff [24] generalized the basic idea as ℓ_p-sampling, which samples an element $i \in \{1, \ldots, n\}$, appearing f_i times in the stream, with probability proportional to ${f_i}^p$ for some specified $p \in \mathbb{R}_{\geqslant}$—in this setting, *Distinct Sampling* would be related to the special case $p = 0$.

Recordinality as a distinct sampling algorithm. One peculiarity of these algorithms, is that the *size* of the sample is a random variable. In the case of *Distinct Sampling*, this random variable oscillates, as mentioned, between $m/2$ and m, with either a binomial distribution or a superimposition of two binomial distributions [20]. This may be less than desirable, as it increases the variance, and thus impacts the accuracy, of any statistic or proportion derived from the sample.

But as it happens, the table of the k largest elements maintained by RECORDINALITY is a uniform sample of exactly k elements of the set underlying the stream[(vi)]. This is easy to see: the presence of an element in RECORDINALITY's cache depends only on its hashed value, which is considered uniformly random.

Distinct samples of tunable size. Furthermore, it seems like an interesting idea to develop sampling algorithms which draw a random sample not of some (near) constant size, but of some size depending on n the *unknown* number of distinct elements for instance $\Theta(\log n)$ or $\Theta(n^\alpha)$ for some (fixed) $\alpha \in (0, 1/2)$. We suggest this seems possible, much like for RECORDINALITY, using other statistics on random permutation with different expected values.

As a brief illustration, we mention the non-standard statistic of *k-substitutions*. Without going into details and providing a rigorous definition, the intuition is as follows. The set of k-records contains some $O(k \log n)$ values on average: of those, some are the k first elements, and others (possibly overlapping) are the k largest values of the permutation. Beyond that nothing is known *a priori* on the remaining k-records other than they were, at some point, part of the locally k largest elements. If it is possible to swap an existing k-record with a larger element that is not itself a k-record, than we do so, and this larger element is then a k-substitution. Of course there may be several such k-substitution.

Figure 3 describes an algorithm which given a stream S, can compute r_k and s_k, respectively the number of k-records and k-substitutions in a stream. Straightforwardly, the size of the cache T is r_k, since any increase in the table size is preceded by an increment to r_k; and as stated by Proposition 1 this happens to be on average $O(k \log n)$. With some thought, it is possible to see that the elements contained

(vi) In fact, RECORDINALITY is related to the limit case of *Distinct Sampling* as noted in [20].

```
fill T with the first k distinct elements of the stream S;  r_k ← k;  s_k ← 0
for all y ∈ S do
    x ← h(y)
    if x ∉ T then
        if x > k-th largest element of T then r_k ← r_k + 1;  T ← T ∪ {x}
        else if x > min(T) then s_k ← s_k + 1;  T ← (T ∪ {x}) \ {min(T)}
    end if
end for
```

Fig. 3: An algorithm to calculate the number of k-records and k-substitutions in a data stream; in the end, the cache T contains a random sample of distinct elements, of size r_k which is expected to be $O(k \log n)$. (For clarity we only show x being added to T, but we actually store a pair (x, y) to be able to recover the actual elements contained in T.)

in T are a uniformly random sample of distinct elements of S, because once again their presence in the cache depends only their hashed value—and not on their position in the stream[(vii)].

This approach differs from taking some threshold θ, such that any element with a hashed value x larger than θ is retained in the sample, because with this approach, one can only obtain a size linear in n—the threshold θ would then need to be dynamically changed, and that brings its lot of complications to ensure uniformity.

This topic is here in a preliminary form, and we hope to develop it in subsequent versions of this paper.

4.2 Usefully combining Recordinality with existing algorithms

It is often the case that estimators for the number of distinct elements are *asymptotically* unbiased. This usually means that they get more accurate as the cardinality to estimate is large; but conversely, they suffer from *nonlinear distortions* for smaller cardinalities.

For instance, the popular *stochastic averaging* technique introduced by Flajolet and Martin [13], which simulates taking many different estimates of a same stream at a fraction of the computational cost, delays the asymptotic regime of an algorithm for well-understood reasons. Several methods have been devised to circumvent this issue. In [21, §3], Lumbroso characterized the initial distortion introduced by stochastic averaging using a Poisson model, and was able to reverse it. In the LogLog family of articles [7, 12], among several others (with an extensive discussion in [6, §5.4]), the idea has been to switch to an auxiliary algorithm, *Linear Counting* [27], when it is detected that the number of distinct elements is small.

This second solution is algorithmically pertinent: stochastic averaging uses an array of buckets, and is distorted when too many buckets are empty; *Linear Counting* uses the number of empty buckets to make its estimate. The data-structure and computations are shared by both estimators, and thus no overhead is required to make use of them both.

Algorithms based on order statistics. But combining an algorithm with *Linear Counting* does not make sense in all cases. Consider Bar-Yossef *et al.*'s first algorithm in [3]: it keeps track of m_k, the k-th

(vii) Note: if instead of T, we were to consider as a sample the set of k-records *without* making substitutions, this would no longer hold true, obviously. For instance, the first k distinct elements of the stream would systematically be included; such a sample would in effect not be uniformly random.

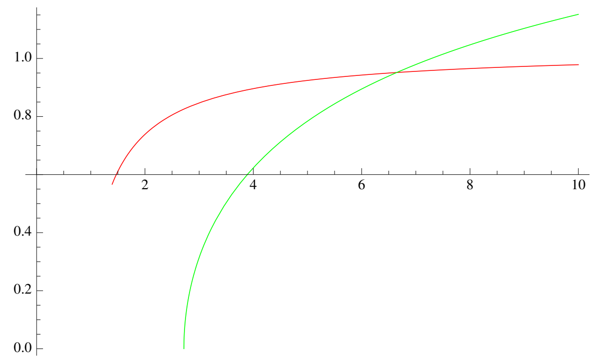

Fig. 4: This plot compares the *theoretical* dispersion of the estimator based on the k-th order statistic (in red) against that of RECORDINALITY using k-records (in green), as a function of the cardinality expressed as a multiple of k. For cardinalities up to about $n = 6k$, RECORDINALITY is less dispersed (and thus leads to more accurate estimates). The y-axis is rescaled to be independent of k.

minimum[(viii)] of the hashed values, and then estimates that $n \sim (k-1)/m_k$.

The data-structure maintained for this algorithm is some kind of a balanced binary tree. *Linear Counting* would in addition require maintaining a separate array. Although this would not change the order of the space and time complexities, it would significantly increase the storage used, doubling it.

On the other hand, Bar-Yossef *et al.*'s order statistic algorithm uses the exact same data-structure as RECORDINALITY, and would only require a small extra $O(\log\log n)$-sized counter to track the number of times the k minima are changed. Figure 4 suggests it would be useful to use RECORDINALITY in conjunction with the estimator based on the k-th order statistics, and to switch to the former for cardinalities n that are smaller than $6k$ (where k is the number of minima stored).

Hybridization. A complementary idea comes from the observation that the *values* of the k smallest elements of a stream are independent from their *position* in the stream and thus, from the number of times the data-structure maintaining the current k smallest elements is updated while scanning the stream.

In other words, the statistic behind Bar-Yossef *et al.*'s first algorithm (the k-th order statistic) and that behind RECORDINALITY (the number of k-records) are *independent*. What if, instead of using one or the either, we used *both*?

We expect the accuracy to improve, if for no other reason than because the standard deviation is sub-additive. If we note Bar-Yossef *et al.*'s first algorithm and RECORDINALITY respectively as $\mathcal{O}_k$ and $\mathcal{R}_k$, we define the hybrid algorithm $\mathcal{H}_{\lambda,k}$ as

$$\mathcal{H}_{\lambda,k} := \lambda\mathcal{O}_k + (1-\lambda)\mathcal{R}_k \qquad \text{with} \qquad \lambda \in (0,1).$$

Because of the aforementioned independence,

$$\mathbb{V}[\mathcal{H}_{\lambda,k}] = \sqrt{\lambda^2\mathbb{V}[\mathcal{O}_k] + (1-\lambda)^2\mathbb{V}[\mathcal{R}_k]}. \tag{2}$$

(viii) There is, of course, a symmetry between the k-th minimum and the k-th maximum.

A suitable value for parameter λ is to be determined, either empirically through simulations, or by plugging in the theoretical variances of the algorithms in (2) and maximizing the resulting function (or some approximation of it).

Finally it is worthwhile to mention that RECORDINALITY can be thus hybridized with any existing cardinality estimation algorithm as, to the best of our knowledge, the statistic it uses is likewise independent with all those previously considered (although, as we have said, it does not always make algorithmic sense).

4.3 Letting go of hash functions: the random-order model

Most cardinality estimation algorithms are based on hash functions. As previously expounded, these have a number of beneficial features, but in this subsection we will only focus on two: first, hash functions chop up and mix the data until it looks *quasi-random*; second, hash functions reduce arbitrary data into *computable values*—either random $(0, 1)$ reals, integers, or random bits. Estimators are then typically just functions which take these values, and output an estimate of the number of distinct elements.

It is noteworthy that these algorithms do not just rely on hash functions for pseudo-randomness, but also for converting the input data to a useful form. Hash functions are thus central, and indeed it is a well-known fact (see for instance [12, §4]) that, in these applications, they generally account for roughly 80% of the run time(ix).

On the other hand, the algorithms we describe in this article, such as RECORDINALITY, never use the values provided by hash functions; instead, because only the *relative ranks* of the hash values are taken into account, the hash functions serve merely to, in a sense (and as noted at the beginning of Section 2), *randomly permute* the input data. But what if the data is *already* a random permutation?

Random-order model. In 1980, Munro and Paterson [25], in the context of selecting the median from a list of unsorted elements, were the first to consider the advantages of assuming all orderings of the stream to be equally likely—in other words, to consider that the input stream is a random permutation.

Initially, the idea did not gain much traction, presumably because this assumption seems even more theoretically unjustifiable than considering hashed values as nearly uniform random variables(x). In 2007, McGregor, in his PhD thesis [22] and related articles, provided the first extensive discussion on the topic and some arguments as to when data streams may be considered random (enough), see [22, §3.1.2]. This allows us to state the following theorem.

Theorem 4. *In the random-order model, the algorithm derived from* RECORDINALITY *by removing the hash function and ranking input elements in lexicographical order, is an unbiased estimator of the number of distinct elements, exhibiting the previously demonstrated properties.*

Initial simulations. Determining the practical applicability of this result would require some serious experimentation and, in all likelihood, would certainly result in algorithms which are not universal: for instance, perhaps the generic algorithm given in Theorem 4 can be tweaked to give accurate estimation for literary English texts, or for some other class of input streams.

Out of sheer curiosity, we ran this version of RECORDINALITY on Shakespeare's tragedies for varying values of k. The results of these simulations are plotted in Figure 5.

(ix) Though in theory hash functions can be calculated extremely fast using hardware, this is, in practice, seldom done.

(x) A consideration which, several decades after it has been empirically validated many times over, keeps getting flak from rigid theoreticians!

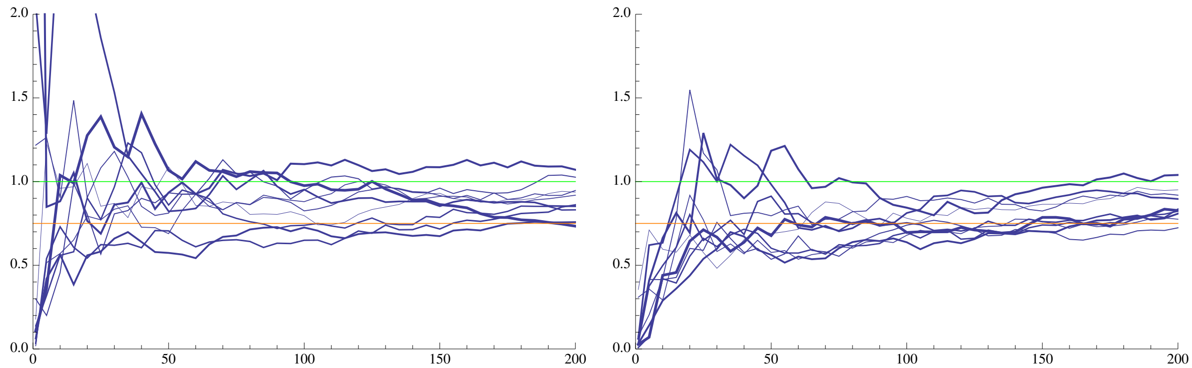

Fig. 5: Here, we determine the accuracy of estimating the number of distinct words (expressed as the ratio of the estimate to the actual number of distinct words) using a straightforward version of RECORDINALITY which forsakes the use of hash function, as a function of k the memory usage. (a) *Left:* each curve corresponds to one of Shakespeare's tragedies, unmodified and processed *as is*; the *thickness* of the curve is proportional to the number of distinct elements contained in the text, which ranges from 2884 to 4725. (b) *Right:* as a control experiment, we also tried using random permutations of the texts.

Several points are striking. First of all, given the fact that Shakespeare's plays cannot reasonably be considered to be random-order streams, it is quite remarkable that the estimates are so accurate: starting around $k = 50$, most estimates fall within a 25% accuracy. As a matter of fact, the algorithm seems to consistently underestimate the number of distinct elements, to such an extent that it might be possible to compensate for this and obtain a better accuracy. The smaller cardinalities (indicated by the thinner lines) might be better estimated than larger cardinalities, but this is far from obvious. Finally, it seems notable that accuracy increases proportionally to the memory used, up until $k = 50$, past which any additional memory seems to be wasted.

A second plot, on the right, gives a second set of experimental results, when instead of the original texts, we use a random permutations of these. Interestingly, the initial gross over-estimations of the previous simulations (which are truncated on the plot) do not seem to occur; while the cut-off in accuracy still seems to be around $k = 50$.

As a final note on this topic, we should of course mention that, for this particular idea, beyond experimental validation it remains to be seen whether the overhead introduced by comparing strings (instead of the integers computed by the hash function) compensates any gain in speed from avoiding hash functions—one possible avenue would be to store the largest elements in a *ternary search tree*. Furthermore, without hash functions, the elements in the cache of RECORDINALITY no longer are a random sample as discussed in Subsection 4.1.

Acknowledgements

The first and the third authors were supported by Project TIN2010-17254 FRADA from the Spanish Ministry of Science and Innovation. The second author was supported by the French ANR Project NT09-432755 BOOLE. The authors are thankful to the referees of AofA 2012 for their benevolent comments, and for providing interesting additional references.

References

[1] Miklós Ajtai, T. S. Jayram, Ravi Kumar, and D. Sivakumar. Approximate counting of inversions in a data stream. In *Proceedings of the Thirty-Fourth ACM Symposium on Theory Of Computing*, pages 370–379. ACM, 2002.

[2] Margaret Archibald and Conrado Martínez. The hiring problem and permutations. In *Proceedings of the Twenty-First International Conference on Formal Power Series and Algebraic Combinatorics*, Discrete Mathematics & Theoretical Computer Science Proceedings, pages 63–76, 2009.

[3] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting Distinct Elements in a Data Stream. In José D. P. Rolim and Salil P. Vadhan, editors, *Randomization and Approximation Techniques (RANDOM)*, pages 1–10. Springer, 2002.

[4] Andrei Z. Broder, Adam Kirsch, Ravi Kumar, Michael Mitzenmacher, Eli Upfal, and Sergei Vassilvitskii. The hiring problem and Lake Wobegon strategies. In *Proceedings of the Nineteenth ACM-SIAM Symposium On Discrete Algorithms*, pages 1184–1193, 2008.

[5] Amit Chakrabarti, Graham Cormode, and Andrew McGregor. Robust lower bounds for communication and stream computation. In *Proceedings of the Fortieth ACM-SIAM Symposium on Theory Of Computing*, pages 641–650. ACM-SIAM, 2008.

[6] Marianne Durand. *Combinatoire analytique et algorithmique des ensembles de données*. PhD thesis, École Polytechnique, France, 2004.

[7] Marianne Durand and Philippe Flajolet. LogLog Counting of Large Cardinalities. In Giuseppe Di Battista and Uri Zwick, editors, *Proceedings of the Eleventh European Symposium on Algorithms*, volume 2832 of *Lecture Notes in Computer Science*, pages 605–617, 2003.

[8] Funda Ergun and Hossein Jowhari. On distance to monotonicity and longest increasing subsequence of a data stream. In *Proceedings of the Nineteenth ACM-SIAM Symposium On Discrete Algorithms*, pages 730–736. Society for Industrial and Applied Mathematics, 2008.

[9] Philippe Flajolet. On adaptive sampling. *Computing*, 34:391–400, 1990.

[10] Philippe Flajolet. Adaptive sampling. In Michiel Hazewinkel, editor, *Encyclopaedia of Mathematics*, volume Supplement I, page 28. Kluwer Academic Publishers, Dordrecht, 1997.

[11] Philippe Flajolet. Counting by coin tossings. In Michael J. Maher, editor, *Proceedings of the Ninth Asian Computing Science Conference*, volume 3321 of *Lecture Notes in Computer Science*, pages 1–12, 2004.

[12] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. HyperLoglog: the analysis of a near-optimal cardinality estimation algorithm. In Philippe Jacquet, editor, *Proceedings of Analysis of Algorithms 2007*, pages 127–146, 2007.

[13] Philippe Flajolet and G. Nigel N. Martin. Probabilistic Counting Algorithms for Data Base Applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985.

[14] Peter R. Freeman. The secretary problem and its extensions: A review. *International Statistical*, 51(2):189–206, 1983.

[15] Phillip B. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. *Proceedings of the International Conference on Very Large Data Bases*, pages 541–550, 2001.

[16] Parikshit Gopalan, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Estimating the sortedness of a data stream. In *Proceedings of the Eighteenth ACM-SIAM Symposium On Discrete Algorithms*, pages 318–327. Society for Industrial and Applied Mathematics, 2007.

[17] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics*. Addison Wesley, Reading, Massachussetts, 2nd edition, 1994.

[18] Ahmed Helmi, Conrado Martínez, and Alois Panholzer. Hiring above the m-th best candidate: a generalization of records in permutations. In *Proceedings of LATIN 2012*, 2012.

[19] Guy Louchard. Probabilistic analysis of adaptive sampling. *Random Structures & Algorithms*, 10(1-2):157–168, 1997.

[20] Jérémie Lumbroso. A full analysis of distinct sampling, and applications. Preprint (2012).

[21] Jérémie Lumbroso. An optimal cardinality estimation algorithm based on order statistics and its full analysis. In *Proceedings of the Twenty-First International Meeting on Probabilistic, Combinatorial, and Asymptotic Methods in the Analysis of Algorithms*, pages 489–504, 2010.

[22] Andrew McGregor. *Processing Data Streams*. PhD thesis, University of Pennsylvania, 2007.

[23] Michael Mitzenmacher and Salil Vadhan. Why simple hash functions work: Exploiting the entropy in a data stream. In *Proceedings of the Nineteenth ACM-SIAM Symposium On Discrete Algorithms*, pages 746–755. Society for Industrial and Applied Mathematics, 2008.

[24] Morteza Monemizadeh and David P. Woodruff. One-pass relative-error L_p-sampling with applications. In *Proceedings of the Twenty-First ACM-SIAM Symposium on Discrete Algorithms*, pages 1143–1160. Society for Industrial and Applied Mathematics, 2010.

[25] J. Ian Munro and Mike S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12(3):315–323, 1980.

[26] S. Muthu Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2):117–236, 2005.

[27] Kyu-Young Whang, Bradley T. Vander-Zanden, and Howard M. Taylor. A linear-time probabilistic counting algorithm for database applications. *ACM Transactions on Database Systems*, 15(2):208–229, 1990.

[28] David Woodruff. Distinct elements is hard even for random data streams, 2008.

A Stochastic Averaging doesn't make sense for Recordinality

Stochastic averaging, as we've said in Subsection 4.2, is a powerful technique to—at its most generic—improve by a factor $1/\sqrt{m}$ the accuracy of a random algorithm which uses hash functions, by efficiently running m concurrent simulations.

This technique is not pertinent to algorithms based on the k-th order statistic, such as [3] or our algorithm RECORDINALITY, because taking the k-th order statistic (as opposed to the minimum, which is the first order statistic) is essentially the same as averaging k minima. In other terms, coupling stochastic averaging with an algorithm based the k-th order statistic is *redundant*.

Even so, we have calculated the accuracy of RECORDINALITY in which we split the stream in m substreams using stochastic averaging (see any of [13, 7, 12, 21] for details), and combined them using an arithmetic mean[xi].

Theorem 5. *When using stochastic average to split the stream into m substreams, the accuracy of estimator $\mathcal{R}_k$, expressed in terms of standard error, asymptotically satisfies*

$$\mathbb{SE}_n[\mathcal{R}_k] \sim \frac{1}{\sqrt{m}}\sqrt{\left(\frac{n}{k\mathrm{e}}\right)^{\frac{1}{k}} - 1},$$

[xi] It was an important development of the LOGLOG family of algorithms that the harmonic mean attenuates the effect of overly large estimates, and provides better estimates; in the case of RECORDINALITY however, the statistic is not prone to over-estimations, but actually, as shown on the distribution given in Subsection 2.2, tends to under-estimate—so the arithmetic mean is preferable.

Theorem 5 makes it clear that, if memory is to be spent, it is always better to increase k than m (and since the total memory usage is km, there is no reason not to favor the former).

B Discardinality: mixing ranks and values

By definition, the set R_k of all k-records necessarily contains the k first elements of the permutation (from an algorithmic perspective, this is when the table is being filled up, see Section 2), and the k largest elements of the permutation (which are contained in the table once the algorithm has scanned the entire permutation). Beyond those, the set R_k may or may not contain any other element of the permutation depending on its order.

This brings us to another statistic on permutations: the *largest non-k-record*; we denote its *value*(xii) by D_k, and its rank by d_k, defined as

$$d_k := \max\left(\{1, \ldots, n\} \setminus R_k\right),$$

and of which the exact distribution can be found in [18, §3.4]. The expected value of these two parameters are related as

$$\mathbb{E}_n[D_k] \sim \frac{\mathbb{E}_n[d_k]}{n+1}. \tag{3}$$

This means that D_k is simply a *mixture* of order statistics. In this sense, DISCARDINALITY, our algorithm based on the largest non-k-record, is very similar in many respects to previous algorithms of the same type [3]—it notably does not exhibit the properties we discussed in Subsections 4.2 and 4.3.

Theorem 6. *Let $D_k \in (0,1)$ be the empirically observed largest value which is not a k-record. Then the estimator $\mathcal{D}_k$ defined by*

$$\mathcal{D}_k := \frac{\gamma_k}{1 - D_k},$$

where γ_k is a corrective factor

$$\gamma_k := k + \sqrt{\pi k/2},$$

is an asymptotically unbiased *estimator of n the number of distinct elements.*

Theorem 7. *The accuracy of the estimator $\mathcal{D}_k$, expressed in terms of standard error, asymptotically satisfies*

$$\mathbb{SE}_n[\mathcal{D}_k] \sim \frac{1.16}{\sqrt{k}}\left(1 + O\left(1/\sqrt{k}\right)\right). \tag{4}$$

This algorithm was originally named after the statistic of "best discarded candidate" (but because we initially could not agree on whether or not to include it in the final version of the paper, it has come to be known to us as... DISCORDINALITY!).

(xii) The value we mention here is the value of the hashed element which is a real in $(0, 1)$—not the value of the original element.