Finding Frequent Elements in Non-bursty Streams*

Rina Panigrahy[1] and Dilys Thomas[2]

[1] Microsoft Research, Mountain View, CA 94043, USA
rina@microsoft.com
[2] Department of Computer Science, Stanford University, CA 94305, USA
dilys@cs.stanford.edu

Abstract. We present an algorithm for finding frequent elements in a stream where the arrivals are not bursty. Depending on the amount of burstiness in the stream our algorithm detects elements with frequency at least t with space between $\tilde{O}(F_1/t^2)$ and $\tilde{O}(F_2/t^2)$ where F_1 and F_2 are the first and the second frequency moments of the stream respectively. The latter space complexity is achieved when the stream is completely bursty; i.e., most elements arrive in contiguous groups, and the former is attained when the arrival order is random. Our space complexity is $\tilde{O}(\alpha F_1/t^2)$ where α is a parameter that captures the burstiness of a stream and lies between 1 and F_2/F_1. A major advantage of our algorithm is that even if the relative frequencies of the different elements is fixed, the space complexity decreases with the length of the stream if the stream is not bursty.

1 Introduction

Finding frequent elements in a stream is a problem that arises in numerous database, networking and data stream applications. The objective is to use a small amount of space while scanning through the stream and detect elements whose frequency exceeds a certain threshold. This problem arises in the context of data mining while computing *Iceberg Queries* [8,2,10] where the objective is to find frequently occurring attribute values amongst database records. The problem of finding *Association Rules* [1] in a dataset also looks for frequently occurring combination of attribute values. This problem is also useful in web traffic analysis for finding frequently occurring queries in web logs [4], hot list analysis [9] and in network traffic measurement for finding heavy network flows [7,16].

Formally, given a stream of n elements the objective is to find all elements with frequency at least t. Let F_i denote the i^{th} frequency moment of the data stream. If the frequency vector of the stream with m ($=F_0$) distinct elements is

* Supported in part by NSF Grant ITR-0331640. This work was also supported in part by TRUST (The Team for Research in Ubiquitous Secure Technology), which receives support from the National Science Foundation (NSF award number CCF-0424422) and the following organizations: Cisco, ESCHER, HP, IBM, Intel, Microsoft, ORNL, Qualcomm, Pirelli, Sun and Symantec.

denoted by $(k_1, k_2, \ldots, k_m)$, i.e., the j^{th} distinct element occurs k_j times, then $F_i = \sum_j k_j^i$.

There are several algorithms to find the frequent elements. Karp *et al.* [12] and Demaine *et al.* [6] provide a deterministic algorithm to exactly find elements with frequency greater than t in a stream of length n. The algorithm requires two passes, linear time and space $O(n/t)$. The first pass is an online algorithm generalizing a well known algorithm to find the majority element [3,15], in a stream. A variant of this algorithm is provided by Manku and Motwani [14]. All these algorithms may produce false positives whose frequencies may be slightly less than t; a second pass is used to filter these out. An algorithm for tracking frequent elements in a dynamic set where elements are being inserted and deleted is presented by Cormode and Muthukrishnan [5].

A randomized algorithm for this problem, called Count-Sketch, by Charikar *et al.* [4] (see also [11]) based on hashing requires $\tilde{O}(F_2/t^2)$ space[1] where, t gives the frequency of the frequent elements of interest and F_2 is the second frequency moment of elements in the stream (where all frequencies are capped at t). Note that F_2/t^2 can be no more than n/t if all frequencies are capped at t. In particular if the non frequent elements appear only a constant number of times this is $\tilde{O}(n/t^2)$. This means that the space complexity depends on the second frequency moment and is large if most elements have a high frequency count. If the elements arrive in random order Demaine *et al.* [6] provide an algorithm that uses space $O(F_1/t^2)$. However this algorithm strongly depends on the assumption that the arrival order is strictly random and does not work in any other case.

The Count-Sketch algorithm, on the other hand works for any arrival pattern while using space $\tilde{O}(F_2/t^2)$ but fails to exploit any randomness in the arrival order; its space complexity does not change whether the elements arrive in bursts or random order. In practice however the arrival pattern in a stream is unpredictable. Although we cannot assume that the arrival order is random, it is unlikely to be entirely bursty; i.e., it is unlikely that the same element arrives in contiguous bursts. In summary there is no single algorithm that adapts to the arrival pattern of the stream.

2 Contributions

We present an algorithm that exploits any absence of burstiness in the arrival pattern. Its space complexity decreases as the arrival pattern becomes less bursty. If all the copies of an element appear in contiguous bursts it takes space $\tilde{O}(F_2/t^2)$ and at the other extreme if elements appear in random order its space complexity gracefully improves to $\tilde{O}(F_1/t^2)$. In general the space requirement lies between $\tilde{O}(F_1/t^2)$ and $\tilde{O}(F_2/t^2)$ depending on the burstiness of the arrivals. Our results

[1] Additional space proportional to the size of the output is required to store the frequent elements once detected. We ignore this from the space complexity of all algorithms.

are especially interesting in light of a recent lower bound [13] of $\Omega(F_2/t^2)$ for this problem. However, the lower bound assumes a bursty arrival pattern.

Our algorithm works by sampling certain elements from the stream and maintaining counters for these elements. The counters are decayed over time and at every increment of a counter all the other counters are decremented by a certain value. The idea of decaying counters has been used in [14] and the decrementing of all counters on an increment has been used in [6,12]. The combination of both these steps along with the sampling gives us the improved result; dropping any of the ingredients gives a higher space complexity of $\Omega(n/t)$. The space complexity of our algorithm is $\tilde{O}(\alpha F_1/t^2)$, where α captures the burstiness of the stream, and lies between 1 and F_2/F_1.

The key advantage of our algorithm is that the space complexity decreases as we are allowed to look over a larger portion of the stream. This means in typical scenarios (where there there stream is not completely bursty) for any given space we can find the frequent elements as long as we are allowed to look at a suitably large portion of the stream. Consider for example a stream of length n where the frequent element occurs in θ fraction of the positions and all the other elements occur in $\theta/2$ fraction of the positions. In this case $\tilde{O}(F_2/t^2) \approx 2/\theta$. In comparison if the burstiness is low enough our algorithm achieves space complexity close to $\tilde{O}(F_1/t^2) \approx 1/(\theta t)$, where the count t of the frequent element is equal to $n\theta$. Since t increases with the length of the stream, this means that the space complexity decreases linearly in the count of the frequent element as the length of the stream increases. Even if the stream is not perfectly random but αF_1 is comparable to $F_{2-\epsilon}$, the space complexity grows as $\tilde{O}(\frac{F_{2-\epsilon}}{t^2}) = 1/(\theta t^{\epsilon})$. Even in this case the space complexity decreases as we use a longer stream.

The burstiness parameter α can be described as follows. Pick an element e at a random position i from the stream and find how many future occurrences of that element cumulatively occur at about the density of the frequent elements. Formally, starting from the location i find the positions of the future occurrences of the element e. Let l_{i1}, l_{i2}, ... l_{ij} be the distances of the first, second, ... jth occurrences of that element after the position i ($l_{i0} = 0$). Observe that the density of occurrence in the segments starting at location i of lengths l_{i1}, l_{i2}, ... l_{ij} are $\frac{1}{l_{i1}}$, $\frac{2}{l_{i2}}$, ... $\frac{j}{l_{ij}}$ and so forth.

Definition 1. *The ρ-burst count $B(i, \rho)$ of an element e at position i in the stream is defined as the maximum number of occurrences of e starting from position i that cumulatively occur at rate at least $\frac{\rho t}{n}$; i.e., $B(i, \rho) = 1 + max\{k | \forall (j \leq k) : \frac{j}{l_{ij}} \geq \frac{\rho t}{n}\}$.*

The additive value of 1 accounts for the occurrence of element e at position i itself.

Definition 2. $F_2(\rho) = \sum_{i=1}^{n} B(i, \rho)$

Let us look at the contribution to $F_2(\rho)$ from a single element with frequency k. Look at the sum of $B(i, \rho)$ where i spans over the k occurrences. For each occurrence, $B(i, \rho)$ is at least 1 and at most k. If all the occurrences appear contiguously then the individual values of $B(i, \rho)$ for these occurrences are $1, 2 \ldots k$ giving a total contribution of $k(k+1)/2$. On the other extreme if all occurrences are far apart with a gap of at least $n/(\rho t)$, then each $B(i, \rho)$ for this element is 1 and the total contribution is k. So $F_1 \leq F_2(\rho) \leq (F_2 + F_1)/2 \leq F_2$.

Definition 3. $\alpha(\rho) = \frac{F_2(\rho)}{F_1}$.

In the computation of $F_2(\rho)$ we consider at most t occurrences of each element. This ensures that an element that occurs more than t times contributes at most a constant to $F_2(\rho)/t^2$. Observe that $F_2(\rho) \geq F_1$; this is because $B(i, \rho)$ is at least 1. $F_2(1)$ approaches F_1 if for most positions in the stream the next occurrence of the element at that position is farther than n/t. This tends to happen as the arrival pattern tends to a random arrival pattern.

We present an algorithm that finds all elements with frequency more than t with high probability using space $\tilde{O}(F_2(\rho)/t^2)$. Our algorithm is allowed to report elements with frequency slightly less than t: it may report elements with frequency at least $(1-\rho-2\epsilon)t$. An element with frequency more than t is reported with probability at least $1-\delta$. The space requirement is $\tilde{O}(\frac{\log(1/\delta) F_2(\rho)}{\epsilon^2 t^2})$. A second pass may be used to filter out the elements with frequency less than t.

3 Algorithm

We will present an algorithm that finds one frequent element with frequency t with probability at least $1 - \delta$ using space $\tilde{O}(\frac{\log(1/\delta) F_2(\rho)}{\epsilon^2 t^2})$. In this section we assume that the value of $F_2(\rho)$ is known a-priori; we will relax this assumption in the next section. For ease of exposition let us focus on one frequent element, with frequency more than t (if it helps the reader may also assume there is only one such frequent element). If there are several frequent elements all of those can be detected by appropriately adjusting the failure probability δ. Note that in the computation of $F_2(\rho)$ the frequency of all elements is capped at t.

Our algorithm essentially maintains counters for certain elements in the stream that are sampled with probability about $\Theta(1/t)$. A counter is incremented if its associated element is found in the stream. Whenever a counter is incremented a fixed value is subtracted from all other counters. A counter is also decayed at a certain rate. Whenever a counter drops below zero it is deleted.

Algorithm:

1. The algorithm maintains a set of counters where each counter is associated with a specific element. If there is a counter associated with an element e we denote that counter value by c_e.
2. While scanning the stream, sample each position at the rate of $1/(\mu t)$, where $\mu = \frac{\log(1/\delta)}{\epsilon}$. When an element e is sampled create a new counter c_e unless a counter already exists for that element. Initialize c_e to 1.

3. For every element in the stream if the element is associated with a counter increment its counter by one and decrement every other counter by d. The value of d will be specified later during our analysis.
4. Decay every counter by $\frac{\rho t}{n}$ every step.
5. Drop a counter if its value drops below zero.
6. Whenever the value of a counter exceeds $(1 - \rho - 2\epsilon)t$ output it and ignore its future occurrences in the stream.

Remark 1. *Although in our algorithm description we perform a decay and a decrement step on each counter separately this can be implemented efficiently in $\tilde{O}(1)$ amortized time per stream entry by keeping a global subtraction counter that is incremented by $\frac{\rho t}{n}$ at every step and by d when an element associated with an active counter is encountered. When a new counter is created, instead of initializing it by 1 we can initialize it by 1 plus the current value in the subtraction counter. The actual value of a counter is its value minus the value of the subtraction counter. When an element with an active counter is found, instead of incrementing it by 1, we increment it by $1 + d$ (since the subtraction counter has also been incremented by d). We maintain the counters in sorted order, and at each step we drop all the counters with value less than the value of the subtraction counter. The amortized time spent per stream element for the drops is constant as the dropping of a counter can be charged to its creation which happens at most once per stream entry.*

The following theorem bounds the space complexity of our algorithm.

Theorem 1. *The algorithm detects a frequent element, with frequency more than t, with probability at least $1 - \delta$ and uses at most $\tilde{O}(\frac{\log(1/\delta)F_2(\rho)}{\epsilon^2 t^2})$ counters.*

We are focusing on detecting one frequent element, say e. Let us look at all the locations in the stream where the frequent element e does not occur. Let $S = \{i_1, i_2, i_3, \ldots, i_p\}$ denote the sampled locations, of the non frequent elements (elements with frequency less than t) sampled at the rate of $1/(\mu t)$. Let q_1, q_2, $\ldots$, q_p denote the burst counts $B(i_1, \rho)$, $B(i_2, \rho)$, $\ldots$, $B(i_p, \rho)$.

Lemma 1. *The counter c_e of an element, e, sampled at position i, can never exceed its burst count $B(i, \rho)$ during its incarnation. (Note that a counter for an element can be created and deleted multiple times).*

PROOF: This is evident from the definition of the burst count, $B(i, \rho)$ and the way the counters are being decayed at rate $\frac{\rho t}{n}$. Whenever the cumulative frequency of the element drops below $\frac{\rho t}{n}$ the counter will drop below zero and hence will be deleted. □

The next lemma computes a high probability upper bound on $\sum_{i=1}^{p} q_i$. Let Q denote this upper bound.

Lemma 2. *The expected value $E[\sum_{i=1}^{p} q_i] \leq \frac{F_2(\rho)}{\mu t}$. With probability at least $(1 - \delta)$ the value of $\sum_{i=1}^{p} q_i \leq O(\frac{F_2(\rho)}{\mu t}) + t\log(\frac{1}{\delta}) = Q$.*

PROOF: Each position i is sampled with probability $1/(\mu t)$ and has burst count $B(i,\rho)$. So, $E[\sum_{i=1}^{p} q_i] = E[1/(\mu t)\sum_{i=1}^{n} B(i,\rho)] = 1/(\mu t) \times F_2^{'} \leq \frac{F_2^{'}}{\mu \times t}$. The high probability part can be proved by an application of Chernoff bounds. Each q_i lies between 0 and t. By scaling them down by a factor of t it lies in $[0,1]$. A standard application of Chernoff bounds gives the result. □

In the algorithm, the decrement value d will be set to $\frac{\mu t}{Q}$ where Q is the high probability upper bound in Lemma 2.

The proof of Theorem 1 is based on the following observations: (1) The frequent element is sampled with high probability. (2) The decay and decrement in steps 3,4 of the algorithm is small so that the counter of the frequent element survives with high value.

Conditioning on sampled locations: We will condition on the sampled locations of all elements other than e, the frequent element of interest. Let S denote the set of sampled elements other than e. We will also condition on the event that $\sum_{i=1}^{p} q_i \leq Q$ which happens with high probability by Lemma 2.

Then we will show that out of the t occurrences of the frequent element, a large fraction of them are good in the sense that if any of them are to be sampled its counter would survive and it would be reported as the frequent element. Conditioned on the sampled locations of the other elements, we will use the following process to identify good locations of the frequent element of interest, say e.

We analyze the following slightly modified algorithm for updating the counter c_e, which differs only in the decrement step.

Modified decrement step for analysis:

1. As before whenever the element e is encountered its counter c_e, if present, is incremented.
2. It is decayed at rate of $\frac{\rho t}{n}$ as before in every step.
3. Whenever an element from S is found a value d is subtracted from the counter c_e if present.

Note that in the above definition the counter operations are almost the same as in the algorithm, except that the decrement operation is allowed to happen even if the counter of an element in S may have expired. Since we are subtracting a larger amount a good element by this definition will always survive in the algorithm.

All the analysis below assumes that we work with this modified algorithm.

Definition 4. Good Occurrence of a frequent element e: *Conditioned on a given set of sampled locations of elements other than e, an occurrence of e is considered good if on initiating the counter c_e at that occurrence the counter remains positive till the end of the algorithm (with the modified decrement step). Any other occurrence of e is called a bad occurrence of e.*

Given the algorithm with the modified decrement step, we can write down with each stream entry the subtraction value that it causes on the counter c_e if present;

this subtraction value is $\frac{\rho t}{n}$ (due to the decay) if the stream entry is not from the set S and $d + \frac{\rho t}{n}$ (due to the decay and the decrement) if it is from the set S. Instead of explicitly performing the decay and the decrement steps for each stream entry we may simply subtract the subtraction value associated with that stream entry from the counter c_e if it is present. The following lemma shows that the total subtraction value of all the stream entries is small.

Lemma 3. *The total subtraction value associated with all the stream entries is at most* $(\epsilon + \rho)t$.

PROOF: The total subtraction value because of the decay is at most ρt. This because the stream is of size n and the decay happens at rate $\frac{\rho t}{n}$.

The total subtraction value from the decrements is no more that dQ. This is because the decrement value of d is applicable every time a an element from S is found, which happens $\sum_{i=1}^{p} q_i$ times. The total decrement value is $d \times (\sum_{i=1}^{p} q_i)$ which by Lemma 2 is at most dQ (recall that we have already conditioned on the high probability event in Lemma 2). Since $d = \frac{\epsilon t}{Q}$ it follows that the total subtraction due to the decrement step is at most ϵt. □

The following lemma shows that there are a large number of good occurrences of the frequent element.

Lemma 4. *Conditioned on* S *there are at least* $(1 - \rho - \epsilon)t$ *good occurrences of element* e.

PROOF: Starting from the first occurrence of e, we check if that occurrence is good or bad. If it is bad, by definition this means there must be a future point in the stream where its counter is dropped below zero due to the subtraction values. We skip all the occurrences of e up to that point and proceed to the next; we will refer to the bad occurrence of e and all the skipped positions as a bad segment (in fact all the occurrences of e in the bad segment can be shown to be bad). This divides the stream into good occurrences and bad segments; any occurrence outside of a bad segment is good. Now we will show that the total number of occurrences in these bad segments is small.

Note that the number of occurrences of e in any bad segment cannot exceed the total subtraction value of the entries in that segment as otherwise the counter c_e initiated at that start of the segment would not have become negative. Since the total subtraction value is at most $(\rho + \epsilon)t$ the total number of occurrences in the bad segments is at most the same value. So the total number of good occurrences is at least $(1 - \rho - \epsilon)t$. □

The following lemma proves that the algorithm reports a frequent element with high probability.

Lemma 5. *With probability at least* $(1-\delta)$ *the frequent element* e *will be sampled and survive with a counter of at least* $(1 - \rho - 2\epsilon)t$.

PROOF: With probability $(1 - \delta)$ the frequent element e is sampled in one of its first $\log(1/\delta) \times \mu t = \epsilon t$ good occurrences. Let s_1 denote the total subtraction value of entries before the sample point, and s_2 denote the total subtraction value of entries after the sample point. The number of bad occurrences before the sample point is at most s_1 (because the number of occurrence in any bad segment is at most the total subtraction value in that bad segment). The total number of occurrences before the sampling point is at most $\epsilon t + s_1$. The number of occurrences after the sampling point is at least $t - (\epsilon t + s_1)$. Further after the sampling point the total subtraction value is s_2. So after the sampling and all the subtractions the counter value in the end is at least $t - (\epsilon t + s_1)$ -s_2 = $t - (\epsilon t + (s_1 + s_2)$. Since $s_1 + s_2 = (\rho + \epsilon)t$ the result follows. □

We are now ready to prove, Theorem 1.

Proof: Let us order the counters by their creation time. We will refer to the number of times a counter has been incremented as its upcount. Since every upcount causes a decrement of all the other counters by value d, the upcount of an active counter has to be at least d times the sum of the upcounts of the newer counters. Let u_i denote the upcount of the i^{th} newest counter in this order (Note that the actual counter values is going to be the upcounts minus the amounts of subtraction due to decaying and decrementing). Since each counter value is non-negative it follows that $u_i > \sum_{j=1}^{i} u_j \times d$. Define $P_i = \sum_{j=1}^{i} u_j$. Hence $P_i > (1+d)P_{i-1}$, so $P_i > (1+d)^i$. Also P_i cannot exceed the total upcount, which is bounded by Q. The total upcount, which is at most $Q = O(\frac{F_2(\rho)}{\mu t}) + t\log(\frac{1}{\delta})$, giving $(1+d)^i < Q$. Approximating $\log(1+d) = d$ for small d, we get $i < (1/d)\log Q = \frac{F_2(\rho)}{\mu\epsilon t^2}\log Q = O(\frac{\log(1/\delta)F_2(\rho)}{\epsilon^2 t^2}\log(\frac{\log(1/\delta)F_2(\rho)}{\epsilon t} + \log(1/\delta))) = \tilde{O}(F_2(\rho)/t^2)$ □

So far we have focussed on only finding one frequent element. By adjusting the high probability parameter δ we can find all elements with frequency more than t. Since there are at most n/t such elements by replacing δ by $\frac{\delta t}{n}$ all such elements can be detected with probability $1-\delta$ by using space $\tilde{O}(\frac{\log(\frac{n}{t\delta})F_2(\rho)}{\epsilon^2 t^2})$.

So far in our analysis we have assumed that all the frequencies are bounded by t. To eliminate this assumption we note that a counter value is never allowed to exceed t as once this happens the element is reported by the algorithm and all its future occurrences are ignored.

4 Algorithm Without Assuming Knowledge of $F_2(\rho)$

We will now show how to run the algorithm without assuming the value of $F_2(\rho)$ is known.

Maintain a global counter q, initialized to zero which is incremented whenever an element with an active counter is encountered in the stream; q tracks the total upcount of all the counters. Note that $q < Q$ with high probability (see Lemma 2). The decrement amount d, which earlier depended on $F_2(\rho)$ is now set to $\frac{\epsilon' \epsilon t}{q \log^{1+\epsilon'}(q)}$.

Algorithm:

1. The algorithm maintains a set of counters, where each counter is associated with a specific element. If there is a counter associated with an element e we denote that counter value by c_e.
2. While scanning the stream, sample each position at the rate of $1/(\mu t)$, where $\mu = \frac{\log(1/\delta)}{\epsilon}$. When an element e is sampled create a new counter, c_e, unless a counter already exists for that element. Initialize c_e to 1.
3. For every element in the stream if the element is associated with a counter, increment its counter by one, decrement every other counter by $d = \frac{\epsilon' \epsilon t}{q \log^{1+\epsilon'}(q)}$ (Here, ϵ' is any small constant), and increment q.
4. Decay every counter by $\frac{\rho t}{n}$ in every step.
5. Drop a counter if its value drops below zero.
6. Whenever the value of a counter exceeds $(1 - \rho - 2\epsilon)t$ output it and ignore its future occurrences in the stream.

The proof of correctness of the improved algorithm is very similar to the previous one except for the following differences.

Lemma 6. *The total subtraction amount due to the decrements is at most ϵt.*

Proof: The subtraction when the upcount is q is $\frac{\epsilon' \epsilon t}{q \log^{1+\epsilon'}(q)}$. The total subtraction is therefore at most $\sum_{q=2}^{\infty} \frac{\epsilon' \epsilon t}{q \log^{1+\epsilon'}(q)} \le \int_2^{\infty} \frac{\epsilon' \epsilon t}{q \log^{1+\epsilon'}(q)} dq \le \int_1^{\infty} \frac{\epsilon' \epsilon t}{x^{1+\epsilon'}} dx \le (\epsilon \times t)$ □

Theorem 2. *The total number of counters used by the improved algorithm is at most $\tilde{O}(\frac{F_2(\rho)}{\mu \epsilon' \epsilon t^2})$.*

Proof: The decrement value d is at least $\frac{\epsilon' \epsilon t}{Q \log^{1+\epsilon'}(Q)}$. Just as in the proof of Theorem 1 the number of counters is at most $(1/d)\log(Q) \le \tilde{O}(\frac{F_2(\rho)}{\mu \epsilon \epsilon' t^2})$ □

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proc. of the Intl. Conf. on Very Large Data Bases, pp. 487–499 (1994)
2. Beyer, K., Ramakrishnan, R.: Bottom-up computation of sparse and iceberg cubes. In: Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, pp. 359–370 (1999)
3. Boyer, R., Moore, S.: MJRTY - a fast majority vote algorithm. In: U. Texas Tech report (1982)
4. Charikar, M., Chen, K., Colton, M.F.: Finding frequent items in data streams. In: Proceedings of the 29th International Colloquium on Automata, Languages and Programming. LNCS, pp. 693–703. Springer, Heidelberg (2002)

5. Cormode, G., Muthukrishnan, S.: What's hot and what's not: Tracking most frequent items dynamically. In: Proc. of the ACM Symp. on Principles of Database Systems (2003)
6. Demaine, E., Ortiz, A.L., Munro, J.I.: Frequency estimation of internet packet streams with limited space. In: Proceedings of the 10th Annual European Symposium on Algorithms, pp. 348–360, Sept (2002)
7. Estan, Verghese, G.: New directions in traffic measurement and accounting. In: ACM SIGCOMM Internet Measurement Workshop, ACM Press, New York (2001)
8. Fang, M., Shivakumar, N., Garcia-Molina, H., Motwani, R., Ullman, J.: Computing iceberg queries efficiently. In: Proc. of 24th Intl. Conf. on Very Large Data Bases, pp. 299–310 (1998)
9. Gibbons, P.B., Matias, Y.: Synopsis data structures for massive data sets. In: DIMACS: Series in Discrete Mathematics and Theoretical Computer Science: Special Issue on Eternal Memory Algorithms and Visualization, vol. A. AMS, Providence, R.I., 39.70 (1999)
10. Han, J., Pei, J., Dong, G., Wang, K.: Efficient computation of iceberg cubes with complex measures. In: Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, pp. 1–12 (2001)
11. Indyk, P., Woodruff, D.: Optimal approximations of the frequency moments of data streams. In: Proceedings of the 37th Annual ACM Symposium on Theory of computing, pp. 22–24. ACM Press, New York (2005)
12. Karp, R., Shenker, S., Papadimitriou, C.: A simple algorithm for finding frequent elements in streams and bags. In: Proc. of the ACM Trans. on Database Systems, pp. 51–55 (2003)
13. Kumar, R., Panigrahy, R.: Instance specific data stream bounds. In: Manuscript (2007)
14. Manku, G.S., Motwani, R.: Approximate frequency counts over data streams. In: Proc. of the Intl. Conf. on Very Large Data Bases (2002)
15. Misra, J., Gries, D.: Finding repeated elements. In: Science of Computer Programming (1982)
16. Pan, R., Breslau, L., Prabhakar, B., Shenker, S.: Approximate fairness through differential dropping. In: ACM SIGCOMM Computer Communication Review, pp. 23–39. ACM Press, New York (2003)