A SIMPLE ALGORITHM FOR FINDING FREQUENT ELEMENTS IN STREAMS AND BAGS

RICHARD M. KARP, CHRISTOS H. PAPADIMITRIOU, AND SCOTT SHENKER

Abstract. We present a simple, exact algorithm for iceberg queries (identifying in a multidet the items with frequency more than a threshold θ) that requires two passes, linear time, and space $1/\theta$.

1. Introduction

In many applications, ranging from network congestion monitoring [5] to data mining [3] and the analysis of web query logs [2], it is often desirable to identify from a very long sequence of symbols (or tuples, or packets) coming from a large alphabet those symbols whose frequency is above a given threshold. Such analysis is sometimes called an "iceberg query" [3, 1] or *hot list* analysis [4]. Since the amount of data is typically huge, it is important that the time and space expended for this analysis be kept to a minimum; and it is sometimes desirable (as in the networking application) that the analysis be done on-line, in one pass over the sequence from left to right (such a sequence is then termed a *stream;* a *bag* is an ordered multiset each element of which can be accessed more than once).

We are thus given a sequence $x = (x_1, \ldots, x_N) \in \Sigma^*$, where Σ is an alphabet with n symbols, and a real number θ (the *threshold*) between zero and one. For the intended class of applications, we can think that $N >> n >> 1/\theta$. Let $f_x(a)$ denote the number of occurrences of a in x. We wish to determine the subset $I(x, \theta) \subset \Sigma$ of symbols defined as $I(x, \theta) = \{a \in \Sigma : f_x(a) > \theta N\}$.

We are interested in several performance criteria for both on-line and off-line algorithms for this problem:

- Amortized time; this is the time required to process all of x, divided by the number N of symbols in x.
- Worst-case time; this applies to on-line algorithms, and is the time required to process a symbol, maximized over all symbols in x.

- Space.

Fang *et al.* [3] address this problem in the bag (off-line) setting, and devise certain sampling, hashing, and hybrid algorithms for it; their algorithms use several passes over the data, and may result in false positives and false negatives. See [4] for a survey of work on this and similar problems.

It is easy to see that the output set $I(x, \theta)$ cannot contain more than $1/\theta$ symbols; however, we point out that determining $I(x, \theta)$ on-line requires $\Omega(n)$ memory. This is of concern because we assume that $n >> 1/\theta$, and memory is of paramount importance in the networking application. We develop a simple on-line algorithm for determining a superset K of $I(x, \theta)$ with at most $1/\theta$ symbols. Our algorithm requires constant worst-case time per symbol and $O(1/\theta)$ space. By a trivial second pass, this yields an off-line algorithm, with the same performance characteristics, for finding precisely $I(x, \theta)$.

2. The Algorithm

We start with the following fact:

Lemma 2.1. $|I(x, \theta)| \leq 1/\theta$

Proof. Otherwise there are more than $\frac{1}{\theta} \cdot \theta N = N$ occurrences of symbols from $I(x, \theta)$ in x. □

We next establish formally the rather obvious fact that no on-line algorithm for computing $I(x, \theta)$ can have memory smaller than n. We leave the precise model of computation unspecified, since the argument is purely information theoretic. For the proof we need that $N > 4n > 16/\theta$.

Proposition 2.2. *Any on-line algorithm for computing $I(x, \theta)$ needs in the worst case $\Omega(n \log(N/n))$ bits.*

Proof. Consider the algorithm crossing the middle of a sequence x, and suppose that no symbol so far has exceeded θN occurrences. Then we claim that the algorithm must at that point have a memory extensive enough to remember the precise tally of each symbol. Because otherwise, if the algorithm reaches the same memory configuration in two computations on two half-sequences with distinct combinations of tallies, say differing in their tally of a, there is a way to complete the two sequences by the same second half so that a

is in $I(x,\theta)$ in one sequence, and not in the other. The algorithm must output the wrong $I(x,\theta)$ in one of the two computations.

The total number of different symbol tallies of strings of length $\lfloor N/2 \rfloor$ is $\binom{\lfloor N/2 \rfloor + n-1}{n-1}$, and it is easy to see using Chernoff bounds that most of these will have no symbol with a tally more than θN. Taking the logarithm establishes the Proposition. $\square$

Hence, on-line algorithms need substantially more than $1/\theta$ space to output the $1/\theta$ or fewer elements of $I(x,\theta)$. We now describe a simple algorithm that identifies a set K of $\lfloor 1/\theta \rfloor$ symbols guaranteed to contain $I(x,\theta)$, using $O(1/\theta)$ memory cells.

Our algorithm generalizes a well known trick for finding a *majority symbol* if it exists —that is, computing $I(x,\theta)$ when $\theta = .5$: Find two different symbols and eliminate them from the sequence. Continue eliminating pairs of distinct symbols, until only one symbol remains (perhaps in many copies). No other symbol can be a majority symbol, because every time a copy of it was eliminated, another symbol was also eliminated. The last symbol can then be tallied to test for majority.

Our algorithm generalizes this idea to $\theta < .5$:

```
x[1]...x[N] is the input sequence of symbols
K is a set of symbols initially empty
count is an array of integers indexed by K
for i:= 1,...,N do
if x[i] is in K then count[x[i]] := count[x[i]] + 1
else  insert x[i] in K, set count[x[i]] := 1,
if |K| > 1/$\theta$ then
for all a in K do
        { count[a] := count[a] - 1,
        if count[a] = 0 then delete a from K $\}\}$
output K}
```

Theorem 2.3. *A superset K of $I(x,\theta)$ with $|K| \leq 1/\theta$ can be computed with $O(1/\theta)$ memory and $O(1)$ operations (including hashing operations) per symbol in the worst case.*

Proof. Consider a symbol a that is not in K at the conclusion of the algorithm. Each occurrence of a was eliminated together with more than $1/\theta - 1$ occurrences of other symbols, and so more than $f_x(a)\theta$ symbols were eliminated in toto. It follows that $f_x(a)\theta < N$, or $f_x < \theta N$. Correctness is immediate.

Implementing K as a hash table, we need $O(1/\theta)$ memory. We also need $O(1)$ *amortized* operations per symbol arrival, since there is a constant number of operations per arrival if no deletions are involved and, if a deletion occurs, the deletion of each symbol can be charged to the time of *its* arrival.

A slightly more sophisticated data structure is needed to make this a worst-case constant time algorithm; we sketch it next. The data structure needs to maintain a set K and a count for each element of K, and support two operations: Increment by one the count of a given element of K, and decrement by one the counts for all elements of K. The set K is maintained again as a hash table, but the counts are maintained implicitly in a linked list $L = (v_1, \ldots, v_c)$, where c is the currently largest count. Each v_j except v_0 has a pointer `symbols` pointing to a doubly linked list of symbols (possibly empty, the set of symbols in K that have a count equal to j). These symbol nodes are pointed to by the corresponding entries of the hash table, and they each point back to v_j. To increment the count of one of these symbols by one, we remove it from v_j's doubly linked list, and we insert it to the one that is one above. If this list does not exist, that is, if $j = c$, extend the list to v_{j+1}. To delete a set of symbol occurrences, we just move the head of L one up from v_1 to v_2 and garbage collect v_1 and the symbols attached to it. It is not hard to see that this data structure correctly maintains K and its counts, and requires a constant amount of time per operation.

As described, the data structure requires space that is proportional to $1/\theta + c$, where c is the largest count; this is a serious problem, for example, if the sequence is a^N. By a simple modification (telescoping a long run of v_i's with null `symbols` by a pointer) the worst-case space bound can be made $O(1/\theta)$. □

Corollary 2.4. *The set $I(x, \theta)$ can be computed with two passes in space $O(1/\theta)$ and $O(N)$ operations, including hashing operations.*

Notice that, since our time bounds include hashing operations they are, strictly speaking, not worst-case bounds, as hashing has a very unfavorable worst-case performance.

References

[1] Kevin S. Beyer and Raghu Ramakrishnan, "Bottom-Up Computation of Sparse and Iceberg CUBEs," *Proceedings of the ACM SIGMOD International Conference on Management of Data,* pages 359-370, June 1999.

[2] Kevin Chen, private communication, December 2001.

[3] Min Fang, Narayanan Shivakumar, Hector Garcia-Molina, Rajeev Motwani, Jeffrey Ullman "Computing Iceberg Queries Efficiently," (1998) Proc. 24th Int. Conf. Very Large Data Bases, VLDB

[4] Phillip B. Gibbons Yossi Matias "Synopsis Data Structures for Massive Data Sets" (1999) DIMACS: Series in Discrete Mathematics and Theoretical Computer Science: Special Issue on Eternal Memory Algorithms and Visualization

[5] Rong Pan, Lee Breslau, Balaji Prabhakar, and Scott Shenker, "Approximate Fairness through Differential Dropping," *preprint*, 2001.