A Simple Algorithm for Finding Frequent Elements in Streams and Bags

RICHARD M. KARP and SCOTT SHENKER
International Computer Science Institute and University of California, Berkeley, California
and
CHRISTOS H. PAPADIMITRIOU
University of California, Berkeley, California

We present a simple, exact algorithm for identifying in a multiset the items with frequency more than a threshold θ. The algorithm requires two passes, linear time, and space $1/\theta$. The first pass is an on-line algorithm, generalizing a well-known algorithm for finding a majority element, for identifying a set of at most $1/\theta$ items that includes, possibly among others, all items with frequency greater than θ.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols—*database management*

General Terms: Algorithms, Measurement, Performance

Additional Key Words and Phrases: Data stream, frequent elements

1. INTRODUCTION

In many applications, ranging from network congestion monitoring [Estan and Varghese 2001; Pan et al. 2001] to data mining [Fang et al. 1998] and the analysis of web query logs [Charikar et al. 2002], 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" [Fang et al. 1998; Beyer and Ramakrishnan 1999] or *hot list analysis* [Gibbons and Matias 1999]. 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

This research was supported by NSF ITR Grant 0081698.
Authors' addresses: International Computer Science Institute, 1947 Center Street, Berkeley, CA 94704; email: christos@cs.berkeley.edu.

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 $N \gg n \gg 1/\theta$. 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 length N of x.

—*Worst-case time*. This applies to on-line algorithms, and is the time required to process an occurrence of a symbol, maximized over all occurrences in x.

—*Number of Passes*.

—*Space*.

Fang et al. [1998] 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. On another front, Alon et al. [1996] show how to use sampling to approximate the *moments* of the frequencies (thus detecting high-frequency items without identifying them). See Gibbons and Matias [1999] 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 (proof: otherwise there are more than $\frac{1}{\theta} \cdot \theta N = N$ occurrences of symbols from $I(x, \theta)$ in x). However, we point out that determining $I(x, \theta)$ on-line requires $\Omega(n)$ memory. This is of concern because we assume that $n \gg 1/\theta$, 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 occurrence and $O(1/\theta)$ space. By a trivial second pass, this yields an off-line algorithm, with the same performance characteristics, for finding precisely $I(x, \theta)$.

Concurrently with the acceptance of this article, we were informed of the unpublished paper [Demaine et al. 2002], which contains essentially the same algorithm presented here.

2. THE ALGORITHM

We start by establishing 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 2.1. *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.

Hence, the algorithm must use memory with at least $\log |K|$ bits, where K is the set of all tally combinations not exceeding θN, that is, the set of all sequences of n integers between 0 and $\theta N - 1$ adding to $\lfloor N/2 \rfloor$. We must derive a lower bound for $|K|$. Each sequence of tallies in K can be visualized as an $n+1$-tuple $\{b_0 = 0, b_1, \ldots, b_{n-1}, b_n = N + n\} \subseteq \{1, 2, \ldots, N + n\}$, (the b_i's being the "boundaries") where the tally of symbol a_i is simply $b_i - b_{i-1} - 1$. Consider the particular sequence alternating between $\lfloor N/2n \rfloor$ and $\lceil N/2n \rceil$. By increasing or decreasing independently each of the b_i's, for $1 \leq i \leq n-1$, by at most $\lfloor N/4n \rfloor$, we obtain $(2\lfloor N/4n \rfloor + 1)^{n-1}$ different sequences in K. Taking logarithms, it follows that the algorithm uses memory of $\Omega(n \log \frac{N}{n})$ bits. □

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 the occurrences of two different symbols and eliminate them from the sequence. Continue eliminating pairs of distinct occurrences, until only one symbol remains (perhaps in many copies). No other symbol can be a majority symbol, because every time an occurrence of it was eliminated, the occurrence of 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
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 2.2. *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 occurrence 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 at least $1/\theta - 1$ occurrences of other symbols, and so more than $f_x(a)/\theta$ occurrences were eliminated altogether. It follows that $f_x(a)/\theta < N$, or $f_x(a) < \theta N$. Correctness is immediate.

Implementing K as a hash table, we need $O(1/\theta)$ memory. We also need $O(1)$ *amortized* operations per occurrence 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 occurrence 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 has a pointer 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_{c+1}. To decrement all counts by one, we just move the head of L one up from v_1 to v_2, we delete v_1, and garbage collect the doubly linked list of symbols attached to it (these are the symbols whose counts were decremented from 1 to 0).

Naturally, garbage collection can be done, as usual, in subsequent rounds, a few cells at a time, thus maintaining constant time per arrival. There is one slight complication: If in one of the next rounds an occurrence of one of the symbols whose count was decremented from 1 to 0 arrives before the corresponding cell of the data structure has been garbage collected, the absence of v_1 will tell the algorithm that the symbol is not currently in K. When the cell corresponding to the symbol is removed, we also remove the symbol from the hash table.

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.

However, as described so far, this 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. Our final modification of the data structure solves this problem. We replace any subsequence of the list L, say $v_i, v_{i+1}, \ldots, v_{i+k}$ for some $k > 1$, where there are no symbols with counts $i + 1, \ldots, i + k - 1$ (that is, the `symbols` pointers out of $v_{i+1}, \ldots, v_{i+k-1}$ are null) with a pointer from v_i to v_{i+k} labeled with a field `length`, whose value is k. The worst-case space bound becomes thus $O(1/\theta)$. □

Once the set K is found in the first pass, with a second pass we can find the tallies of all symbols in K, and then delete from K the symbols with tally less than θN. The resulting set is $I(x, \theta)$.

COROLLARY 2.3. *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, in theory, a very unfavorable worst-case performance.

ACKNOWLEDGMENTS

We want to thank the four reviewers for their constructive and helpful comments.

REFERENCES

ALON, N., MATIAS, Y., AND SZEGEDY, M. 1996. The space complexity of approximating the frequency moments. In *Proceedings of the ACM Symposium on Theory of Computing*. ACM, New York.

BEYER, K. S. AND RAMAKRISHNAN, R. 1999. Bottom-up computation of sparse and iceberg cubes. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, New York.

CHARIKAR, M., CHEN, K., AND FARACH-COLTON, M. 2002. Finding frequent items in data streams. In *ICALP 2002*. Lecture Notes in Computer Science, Springer-Verlag, Heidelberg, Germany, 693–703.

DEMAINE, E. D., MUNRO, J. I., AND LOPEZ-ORTIZ, A. 2002. Frequency estimation of internet packet streams with limited space. In *European Symposium on Algorithms (ESA)*. Lecture Notes in Computer Science, Springer-Verlag, Heidelberg, Germany.

ESTAN, C. AND VARGHESE, G. 2001. New directions in traffic measurement and accounting. In *Proceedings of the SIGCOMM Internet Measurement Workshop*. ACM, New York.

FANG, M., SHIVAKUMAR, N., GARCIA-MOLINA, H., MOTWANI, R., AND ULLMAN, J. 1998. Computing iceberg queries efficiently. In *Proceedings of the 24th International Conference on Very Large Data Bases, VLDB*. Morgan-Kaufmann, San Mateo, Calif., 299–310.

GIBBONS, P. B. AND MATIAS, Y. 1999. 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.

PAN, R., BRESLAU, L., PRABHAKAR, B., AND SHENKER, S. Approximate fairness through differential dropping. preprint, 2001.

Received December 2001; revised June 2002; accepted September 2002