| | |
|---|---|
| Title: | Misra-Gries Summaries |
| Name: | Graham Cormode[1] |
| Affil./Addr. | Department of Computer Science, University of Warwick, Coventry, UK |
| Keywords: | streaming algorithms; frequent items; approximate counting |
| SumOriWork: | 1982; Misra, Gries |

# Misra-Gries Summaries

Graham Cormode[1]
Department of Computer Science, University of Warwick, Coventry, UK

## Years aud Authors of Summarized Original Work

1982; Misra, Gries

## Keywords

streaming algorithms; frequent items; approximate counting

## Problem Definition

The frequent items problem is to process a stream of items and find all items occurring more than a given fraction of the time. It is one of the most heavily studied problems in data stream algorithms, dating back to the 1980s. Many applications rely directly or indirectly on finding the frequent items, and implementations are in use in large scale industrial systems. Informally, given a sequence of items, the problem is simply to find those items which occur most frequently. Typically, this is formalized as finding all items whose frequency exceeds a specified fraction of the total number of items. Variations arise when the items are given weights, and further when these weights can also be negative.

**Definition 1.** *Given a stream $\mathcal{S}$ of $n$ items $t_1 \ldots t_n$, the frequency of an item $i$ is $f_i = |\{j|t_j = i\}|$. The exact $\phi$-frequent items comprise the set $\{i|f_i > \phi n\}$.*

***Example.*** The stream $\mathcal{S} = (a, b, a, c, c, a, b, d)$ has $f_a = 3, f_b = 2, f_c = 2, f_d = 1$. For $\phi = 0.2$, the frequent items are $a, b$ and $c$.

A streaming algorithm which solves this problem must use a linear amount of space, even for large values of $\phi$: Given an algorithm that claims to solve this problem, we could insert a set $S$ of $N$ items, where every item has frequency 1. Then, we could also insert $N$ copies of item $i$. If $i$ is then reported as a frequent item (occurring more than 50% of the time) then $i \in S$, else $i \notin S$. Consequently, since set membership requires $\Omega(N)$ space, $\Omega(N)$ space is also required to solve the frequent items problem. Instead, an approximate version is defined based on a tolerance for error $\epsilon$.

**Algorithm 0.1:** MISRA-GRIES($k$)

$n \leftarrow 0; T \leftarrow \emptyset;$
**for each** $i$ :
**do**
- $n \leftarrow n + 1;$
- **if** $i \in T$
  - **then** $c_i \leftarrow c_i + 1;$
  - **else if** $|T| < k - 1$
    - **then** $T \leftarrow T \cup \{i\}; c_i \leftarrow 1;$
  - **else for all** $j \in T$
    - **do** $c_j \leftarrow c_j - 1;$
      - **if** $c_j = 0$
        - **then** $T \leftarrow T \backslash \{j\};$

**Definition 2.** *Given a stream $\mathcal{S}$ of $n$ items, the $\epsilon$-approximate frequent items problem is to return a set of items $F$ so that for all items $i \in F$, $f_i > (\phi - \epsilon)n$, and there is no $i \notin F$ such that $f_i > \phi n$.*

Since the exact ($\epsilon = 0$) frequent items problem is hard in general, we will use "frequent items" or "the frequent items problem" to refer to the $\epsilon$-approximate frequent items problem. A related problem is to estimate the frequency of items on demand:

**Definition 3.** *Given a stream $\mathcal{S}$ of $n$ items defining frequencies $f_i$ as above, the frequency estimation problem is to process a stream so that, given any $i$, an $\hat{f}_i$ is returned satisfying $\hat{f}_i \leq f_i \leq \hat{f}_i + \epsilon n$.*

# Key Results

The problem of frequent items dates back at least to a problem first studied by Moore in 1980 [5]. It was published as a 'problem' in the Journal of Algorithms in the June 1981 issue [17], to determine if there a majority choice in a list of $n$ votes.

## Preliminaries: the Majority algorithm

In addition to posing the majority question as a problem, Moore also invented the MAJORITY algorithm along with Boyer in 1980, described in a technical report from early 1981 [4]. A similar solution with proof of the optimal number of comparisons was provided by Fischer and Salzburg [9]. MAJORITY can be stated as follows: store the first item and a counter, initialized to 1. For each subsequent item, if it is the same as the currently stored item, increment the counter. If it differs, and the counter is zero, then store the new item and set the counter to 1; else, decrement the counter. After processing all items, the algorithm guarantees that if there is a majority vote, then it must be the item stored by the algorithm. The correctness of this algorithm is based on a pairing argument: if every non-majority item is paired with a majority item, then there should still remain an excess of majority items. Although not posed as a streaming problem, the algorithm has a streaming flavor: it takes only one pass through the input (which can be ordered arbitrarily) to find a majority item. To verify that the stored item really is a majority, a second pass is needed to simply count the true number of occurrences of the stored item.

## Misra-Gries summary

The Misra-Gries summary is a simple algorithm that solves the frequent items problem. It can be viewed as a generalization of MAJORITY to track multiple frequent items.

Instead of keeping a single counter and item from the input, the MISRA-GRIES summary stores $k-1$ (item,counter) pairs. The natural generalization of MAJORITY is to compare each new item against the stored items $T$, and increment the corresponding counter if it is amongst them. Else, if there is some counter with count zero, it is allocated to the new item, and the counter set to 1. If all $k-1$ counters are allocated to distinct items, then all are decremented by 1. A grouping argument is used to argue that any item which occurs more than $n/k$ times must be stored by the algorithm when it terminates. Pseudocode to illustrate this algorithm is given in Algorithm 0.1, making use of set notation to represent the operations on the set of stored items $T$: items are added and removed from this set using set union and set subtraction respectively, and we allow ranging over the members of this set (thus implementations will have to choose appropriate data structures which allow the efficient realization of these operations). We also assume that each item $j$ stored in $T$ has an associated counter $c_j$. For items not stored in $T$, then $c_j$ is defined as 0 and does not need to be explicitly stored.

This $n/k$ generalization was first proposed by Misra and Gries [16]. The time cost of the algorithm is dominated by the $O(1)$ dictionary operations per update, and the cost of decrementing counts. Misra and Gries use a balanced search tree, and argue that the decrement cost is amortized $O(1)$; Karp *et al.* propose a hash table to implement the dictionary [11]; and Demaine *et al.* show how the cost of decrementing can be made worst case $O(1)$ by representing the counts using offsets and maintaining multiple linked lists [8].

The original paper focused on retrieving the set of frequent items, and did not study how accurately the associated counts were. Bose *et al.*[3] observed that executing this algorithm with $k = 1/\epsilon$ ensures that the count associated with each item on termination is at most $\epsilon n$ below the true value. The bounds on the accuracy of the structure were tightened by Berinde *et al.* to show that the error depends only on the "tail": the total weight of items outside the top-$k$ most frequent, rather than the total weight of all items [2]. This gives a stronger accuracy guarantee when the input distribution is skewed, for example if the frequencies follow a Zipfian distribtuion. They also show that the algorithm can be altered to tolerate updates with weights, rather than assuming that each item has equal, unit weight.

A similar data structure called SPACESAVING was introduced by Metwally *et al.* [15]. This structure also maintains a set of items and counters, but follows a different set of update rules. Recently, it was shown that the SPACESAVING structure is isomorphic to MISRAGRIES: the state of both structures can be placed in correspondence as each update arrives [1]. The different representations reflect that SPACESAVING maintains an upper bound on the count of stored items, while MISRAGRIES keeps a lower bound. In studies, the upper bound tends to be closer to the true count, but it is straightforward to switch between the two representations.

Moreover, Agarwal *et al.* [1] showed that the MISRA-GRIES summary is *mergable*. That is, two summaries of different inputs of size $k$ can be combined together to obtain a new summary of size $k$ that summarizes the union of the two inputs. This merging can be done repeatedly, to summarize arbitrarily many inputs in arbitrary configurations. This allows the summary to be used in distributed and parallel environments.

Lastly, the concept behind the algorithm, of tracking information on $k$ representative elements has inspired work in other settings. Liberty [12] showed how this can be used to track an approximation to the best $k$-rank summary of a matrix, using

$k$ rows. This was extended by Ghashami and Phillips [10] to offer better accuracy by keeping more rows.

## Applications

The question of tracking approximate counts for a large number of possible objects arises in a number of settings. Many applications have arisen in the context of the Internet, such as tracking the most popular source, destinations, or source-destination pairs (those with the highest amount of traffic); or tracking the most popular objects, such as the most popular queries to a search engine, or the most popular pieces of content in a large content host. It forms the basis of other problems, such as finding the frequent itemsets within a stream of transactions: those subsets of items which occur as a susbet of many transactions. Solutions to this problem have used ideas similar to the count and prune strategy of the Misra-Gries summary to find approximate frequent itemsets [14]. Finding approximate counts of items is also needd within other stream algorithms, such as approximating the entropy of a stream [6].

## Open Problems

None is reported.

## Experimental Results

There have been a number of experimental studies of Misra-Gries and related algorithms, for a variety of computing models. These have shown that the algorithm is accurate and fast to execute [7; 13].

## URLs to Code and Data Sets

Code for this algorithm is widely available:

`http://www.cs.rutgers.edu/~muthu/massdal-code-index.html`
`http://hadjieleftheriou.com/sketches/index.html`
`https://github.com/cpnielsen/twittertrends`

## Cross-References

## Recommended Reading

1. Agarwal P, Cormode G, Huang Z, Phillips J, Wei Z, Yi K (2012) Mergeable summaries. In: ACM Principles of Database Systems
2. Berinde R, Cormode G, Indyk P, Strauss M (2009) Space-optimal heavy hitters with strong error bounds. In: ACM Principles of Database Systems
3. Bose P, Kranakis E, Morin P, Tang Y (2003) Bounds for frequency estimation of packet streams. In: SIROCCO
4. Boyer B, Moore J (1981) A fast majority vote algorithm. Tech. Rep. ICSCA-CMP-32, Institute for Computer Science, University of Texas

5. Boyer RS, Moore JS (1991) MJRTY - a fast majority vote algorithm. In: Automated Reasoning: Essays in Honor of Woody Bledsoe, Automated Reasoning Series, Kluwer Academic Publishers, pp 105–117
6. Chakrabarti A, Cormode G, McGregor A (2007) A near-optimal algorithm for computing the entropy of a stream. In: ACM-SIAM Symposium on Discrete Algorithms
7. Cormode G, Hadjieleftheriou M (2009) Finding the frequent items in streams of data. Communications of the ACM 52(10):97–105
8. Demaine E, López-Ortiz A, Munro JI (2002) Frequency estimation of internet packet streams with limited space. In: European Symposium on Algorithms (ESA)
9. Fischer M, Salzburg S (1982) Finding a majority among $n$ votes: Solution to problem 81-5. Journal of Algorithms 3(4):376–379
10. Ghashami M, Phillips JM (2014) Relative errors for deterministic low-rank matrix approximations. In: ACM-SIAM Symposium on Discrete Algorithms, pp 707–717
11. Karp R, Papadimitriou C, Shenker S (2003) A simple algorithm for finding frequent elements in sets and bags. ACM Transactions on Database Systems 28:51–55
12. Liberty E (2013) Simple and deterministic matrix sketching. In: ACM SIGKDD, pp 581–588
13. Manerikar N, Palpanas T (2009) Frequent items in streaming data: An experimental evaluation of the state-of-the-art. Data Knowledge Engineering 68(4):415–430
14. Manku G, Motwani R (2002) Approximate frequency counts over data streams. In: International Conference on Very Large Data Bases, pp 346–357
15. Metwally A, Agrawal D, Abbadi AE (2005) Efficient computation of frequent and top-$k$ elements in data streams. In: International Conference on Database Theory
16. Misra J, Gries D (1982) Finding repeated elements. Science of Computer Programming 2:143–152
17. Moore J (1981) Problem 81-5. Journal of Algorithms 2:208–209