Merging Frequent Summaries

M. Cafaro, M. Pulimeno

University of Salento, Italy
{massimo.cafaro, marco.pulimeno}@unisalento.it

Abstract. Recently, an algorithm for merging counter-based data summaries which are the output of the *Frequent* algorithm (Frequent summaries) has been proposed by Agarwal et al. In this paper, we present a new algorithm for merging Frequent summaries. Our algorithm is fast and simple to implement, and retains the same computational complexity of the algorithm presented by Agarwal et al. while providing better frequency estimation.

1 Introduction

In 2011, we presented an algorithm [1] for merging in parallel counter-based data summaries which are the output of the *Frequent* [2] algorithm. Recently, we also designed a parallel algorithm for merging Space Saving summaries [3] and an algorithm for mining frequent items in the time fading model [4]. In 2012, a new algorithm for merging counter-based data summaries which are the output of the *Frequent* algorithm has been proposed by Agarwal et al. [5].

Given a data set A of n items t_1, t_2, ..., t_n, the frequency of an item i is $f_i = |\{j \,|t_j = i\}|$. Let $\tilde{f}_i$ be the frequency reported by the algorithm for item i. The absolute error of item i is defined as the difference $|f_i - \tilde{f}_i|$. The (absolute) *total error* is then the sum of the absolute errors related to the items reported by an algorithm.

In this paper, we present a new algorithm for merging Frequent summaries (based on our previous algorithm) which is fast and simple to implement, and retains the same computational complexity of the algorithm presented in [5] while providing better frequency estimation. We briefly recall notations and definitions used in the sequel.

Definition 1. *Given a* multiset $\mathcal{N}$*, with* $|\mathcal{N}| = n$*, and* $2 \leq k \leq n$*, a* frequent item *(or k–majority element) is an element* $x \in N$ *whose* frequency $f_{\mathcal{N}}(x)$ *is such that* $f_{\mathcal{N}}(x) \geq \lfloor \frac{n}{k} \rfloor + 1$*. The frequent items (or k–majority) problem takes as input an array* $\mathcal{N}$ *of n numbers (a multiset), and requires as output the set* $S = \left\{x \in N : f_{\mathcal{N}}(x) \geq \lfloor \frac{n}{k} \rfloor + 1\right\}$

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 280–285 published in CEUR Workshop Proceedins Vol-1720 at http://ceur-ws.org/Vol-1720

Definition 2. *Merged summary.*
Given k, the k-majority parameter, let $\mathcal{A}_1$ and $\mathcal{A}_2$ be respectively the data sets from which the data summaries $\mathcal{S}_1$ and $\mathcal{S}_2$ are derived by an application of the Frequent algorithm, and let $n = |\mathcal{A}_1| + |\mathcal{A}_2|$. The merged summary $\mathcal{M}$ *is the multiset which contains all of the k-majority elements, i.e., all of the elements whose frequency in $\mathcal{A}_1 \biguplus \mathcal{A}_2$ is greater than or equal to $\lfloor \frac{n}{k} \rfloor + 1$. Moreover, all of the guarantees assured by Frequent on its output continue to hold for the summary $\mathcal{M}$ with reference to the input $\mathcal{A}_1 \biguplus \mathcal{A}_2$.*

Definition 3. 2*-way merging problem.*
Input: k, the k-majority parameter; two summaries $\mathcal{S}_1$ and $\mathcal{S}_2$ derived by an application of the Frequent algorithm.
Output: The merged summary $\mathcal{M}$*.*

The paper is organized as follows. We present in Section 3 our algorithm. In Section 4, the proposed algorithm is analyzed in terms of correctness, computational complexity and total error committed. Full details and proofs will appear in a forthcoming extended version. We draw our conclusions in Section 5.

2 Related Work

In [5], Agarwal et al. introduced an algorithm for merging two data summaries $\mathcal{S}_1$ and $\mathcal{S}_2$ outputted by the *Frequent* algorithm. In the following, given a counter C_i, the notation C_i^e refers to the item monitored by the i–th counter, whilst C_i^f refers to its estimated frequency.

Algorithm 1 Merging Algorithm by Agarwal et al.

Require: $\mathcal{S}_1$; an array of counters; $\mathcal{S}_2$; an array of counters; k, k-majority parameter (the number of counters is $k-1$);
Ensure: an array containing k–majority candidate elements

```
1: procedure MERGE(S1, S2, k)                    ▷ a merged summary of S1 and S2
2:     S ← COMBINE(S1, S2);
3:     if S.nz > k − 1 then                      ▷ prune counters in S
4:         for i = k to 2k − 2 do
5:             C_i^f ← C_i^f − C_{k−1}^f;
6:         end for
7:     end if
8:     return S[k ... (2k − 2)];                 ▷ return the last k − 1 counters
9: end procedure
```

The algorithm works as follows. It starts combining as usual the two data summaries, by adding the frequencies of counters monitoring the same item. This could entail, for Frequent summaries, the use of up to $2k-2$ counters in the worst case, when $\mathcal{S}_1$ and $\mathcal{S}_2$ share no item. Let $\mathcal{S}$ be the combined summary, and $\mathcal{S}.nz$ the number of nonzero counters. Moreover, assume, without loss of generality,

that the total number of counters in $\mathcal{S}$, denoted by $\mathcal{S}.length$, is exactly $2k-2$ and they are stored in sorted ascending order. Indeed, it is always possible to pad the first $\mathcal{S}.length - \mathcal{S}.nz$ positions in $\mathcal{S}$ with dummy counters whose frequency is zero.

If $\mathcal{S}.nz \leq k-1$ the algorithm returns the last $k-1$ counters of $\mathcal{S}$. Otherwise, a pruning operation is required. Then, the algorithm subtracts from the last $k-1$ counters the frequency of the C_{k-1}-th counter and returns the pruned counters. The algorithm requires in the worst case time linear in the total number of counters, i.e., $O(k)$ if implemented as described in [5] using an hash table.

We now analyze the total error committed by this algorithm. Clearly, combining the two data summaries can be done without any additional error. However, the pruning operation occurring when the size of $\mathcal{S}$ is greater than $k-1$ induces a total error $E_T = (k-1)C^f_{k-1}$, i.e., $k-1$ times the frequency of the C_{k-1}-th counter in $\mathcal{S}$. The authors proved that the additional error introduced by the merge is within the error bound guaranteed by Frequent.

3 New Merging Algorithm

In this Section we present our algorithm, shown in pseudo-code as Algorithm 2 for merging two Frequent summaries.

Algorithm 2 Merging Algorithm for Frequent summaries.

Require: $\mathcal{S}_1$; an array of counters; $\mathcal{S}_2$; an array of counters; k, k-majority parameter (the number of counters is $k-1$);
Ensure: an array containing k–majority candidate elements

```
1: procedure MERGE(S1, S2, k)                          ▷ a merged summary of S1 and S2
2:     S ← COMBINE(S1, S2);
3:     if S.nz ≤ k − 1 then
4:         return S[k ... (2k − 2)];                    ▷ return the last k − 1 counters
5:     else▷ build the merged summary M, consisting of counters monitoring item e_i
       with frequency f_i, i = 1, ..., k − 1, as follows:
6:         e_1 ← C^e_k
7:         f_1 ← C^f_k − C^f_{k−1};
8:         M[1] ← (e_1, f_1);
9:         for i = 2 to k − 1 do
10:            e_i ← C^e_{k−1+i}
11:            f_i ← C^f_{k−1+i} − C^f_{k−1} + C^f_{i−1};
12:            M[i] ← (e_i, f_i);
13:        end for
14:        return M;
15:    end if
16: end procedure
```

Algorithm 2 starts by combining the two input summaries into a combined summary $\mathcal{S}$. Then, if the number of nonzero counters in $\mathcal{S}$ is less than or equal

to $k-1$, the algorithm returns as merged summary the last $k-1$ counters of $\mathcal{S}$. Otherwise, the last $k-1$ counters are first updated using exact closed-form equations and then reported as output. Actually, these determining equations produce the same merged summary that we would obtain applying the Frequent algorithm to the combined summary $\mathcal{S}$, a procedure we described and proved to be correct in [1]. Indeed, in [1] a slightly modified version of Frequent is used on $\mathcal{S}$, in which the update step is carefully modified so that each update still requires $O(1)$ time in the worst case. These modifications simply consist in one-shot updates: for each item in $\mathcal{S}$ to be processed, we increment one-shot the counter in charge of monitoring it by a number of occurrences equal to the item's counter in $\mathcal{S}$. In the next Section, we shall show the determining equations, state the correctness of the algorithm and analyze its complexity in the worst case and the total error committed. The main result of the paper is the proof that the following properties hold for our algorithm: (i) it retains the same complexity of the Algorithm proposed by Agarwal et al [5], and (ii) its total error committed is smaller or equal.

4 Analysis

4.1 Complexity Analysis

Lemma 1. *The computational complexity of our Algorithm 2 is $O(k)$ in the worst case.*

4.2 Correctness of Algorithm 2

By construction, the combine step producing $\mathcal{S}$ preserves the frequent items in $S_1 \biguplus S_2$ since no item is discarded and no occurrences are lost. Therefore, it suffices to show that our closed-form equations produce the same merged summary which would be outputted by an application of Frequent (the one-shot update version) to the combined summary. Let $\mathcal{S}.length = 2k-2$ and assume $k \leq \mathcal{S}.nz \leq 2k-2$. We denote by C_j the j–th counter in $\mathcal{S}$, $j = 1, \ldots, 2k-2$, and by e_j^i and m_j^i, respectively, the item monitored by the j–th counter of Frequent (denoted as M_j) and its value at the end of the i–th update step, $i = 0, \ldots, k-1$ and $j = 1, \ldots, k-1$. We define $e_j^0 = C_j^e$ and $m_j^0 = C_j^f$, $j = 1, \ldots, k-1$. Indeed, the step zero reflects the situation in which we have already filled the first $k-1$ counters in the Frequent data structure with the corresponding initial $k-1$ counters in $\mathcal{S}$. This is correct owing to the following facts: (i) the counters in $\mathcal{S}$ are stored in ascending sorted order with respect to the frequencies, (ii) the items in $\mathcal{S}$ are distinct and (iii) Frequent works by assigning an item which is not currently monitored to a new counter if available and maintaining the ascending sorted order with respect to the frequencies.

Theorem 1. *For each update step $i = 1, \ldots, k-1$ and position $j = 1, \ldots, k-1$, the values e_j^i and m_j^i can be defined as follows:*

$$e_j^i = C_{i+j}^e \; j = 1, \ldots, k-1 \tag{1}$$

$$m_j^i = \begin{cases} C_{i+j}^f - C_i^f & j = 1, \ldots, k-i \\ C_{i+j}^f - C_i^f + C_{i+j-k}^f & j = k-i+1, \ldots, k-1 \end{cases} \tag{2}$$

4.3 Total Error Committed By Algorithm 2

In what follows, we assume that after the combine step we are left with a data summary $\mathcal{S}$ consisting of more than $k-1$ nonzero counters. Otherwise, both algorithms do not commit any additional error, owing to the fact that the combine step obviously does not incur any error. Therefore, assuming that $\mathcal{S}$ consists of more than $k-1$ nonzero counters, the total error committed by our algorithm is the total error committed by Frequent when applied to $\mathcal{S}$. The counters' frequencies at the end of the $(k-1)$–th update step are m_j^{k-1}, $j = 1, \ldots, k-1$. Consequently, since Frequent underestimates the frequencies, the total error committed is

$$E_T = \sum_{j=1}^{k-1} C_{k-1+j}^f - m_j^{k-1} \tag{3}$$

We claim that the total error committed by Algorithm 2 is less than or equal to the total error committed by algorithm [5].

Theorem 2. *The following inequality holds*

$$\sum_{j=1}^{k-1} (C_{k-1+j}^f - m_j^{k-1}) \leq (k-1) C_{k-1}^f. \tag{4}$$

5 Conclusions

In this paper we have introduced a new algorithm for merging Frequent summaries and compared it to the algorithm proposed by Agarwal et al. from a theoretical perspective. Our algorithm uses exact closed-form equations for determining the outputs; we have shown that it retains the same computational complexity, whilst providing better frequency estimation. Future work includes designing and carrying out several numerical experiments in order to compare the two algorithms we have discussed from a quantitative perspective.

References

1. Cafaro, M., Tempesta, P.: Finding frequent items in parallel. Concurr. Comput. : Pract. Exper. **23** (2011) 1774–1788
2. Demaine, E.D., López-Ortiz, A., Munro, J.I.: Frequency estimation of internet packet streams with limited space. In: ESA. (2002) 348–360
3. Cafaro, M., Pulimeno, M., Tempesta, P.: A parallel space saving algorithm for frequent items and the hurwitz zeta distribution. Information Sciences **329** (2016) 1 – 19

4. Cafaro, M., Pulimeno, M., Epicoco, I., Aloisio, G.: Mining frequent items in the time fading model. Information Sciences **370 - 371** (2016) 221 – 238
5. Agarwal, P.K., Cormode, G., Huang, Z., Phillips, J., Wei, Z., Yi, K.: Mergeable summaries. In: Proceedings of the 31st Symposium on Principles of Database Systems. PODS '12, New York, NY, USA, ACM (2012) 23–34