Final Project

Tailai Zhang & XXXXXXX

May 11, 2023

Please note that this document has been modified. Contents from the other contributor have been removed to reflect my personal contribution to the write-up.

1 Introduction

In data analysis and machine learning, identifying the most frequent or influential elements (known as Heavy-Hitters) in a dataset is a fundamental task. The primary motivation behind the Heavy Hitters algorithms is the need of processing and analyzing large-scale data in real-time with limited space. One intuitive approach is to keep a count for every unique item. However, this approach is not scalable nor efficient enough in applications that need to handle massive datasets, such as monitoring network traffic in routers. Therefore, specialized algorithms are needed to identify such Heavy Hitters(HH) in an efficient manner, without incurring excessive load on the memory.

Misra–Gries algorithm and Count-Sketch are among the algorithms covered in this class for Heavy-Hitters. The aim of this project is to implement algorithms developed by Arnab et al. [BDW16] and Vladimir et al. [Bra+15], which claimed to be improvements for Misra–Gries and Count-Sketch respectively. Our work will mainly focus on comprehending both algorithms, implementing them, and comparing these algorithms with those covered in our class. By examining the advantages and limitations of these algorithms, hopefully we can get a deeper understanding of how to effectively use them for analyzing datasets and extracting meaningful insights.

2 l_1-Heavy-Hitters

Problem Definition: In the l_1-(ϵ, ϕ)-Heavy Hitters problem, we are given parameters $0 < \epsilon \leq 0.5$ and $2\epsilon \leq \phi \leq 1$, and a stream $a_1, a_2, ..., a_m$ such that item $a_i \in \{1, 2, ..., n\}$. Let f_i be the frequency of item i. The algorithm solves l_1-Heavy-Hitters if, in one pass of the stream, it outputs a set $S \subseteq \{1, 2, ..., n\}$ s.t. if $f_i \geq \phi m$ then $i \in S$, while if $f_i \leq (\phi - \epsilon)m$ then $i \notin S$.

2.1 Motivation

To understand the motivation of the algorithm proposed by Arnab et al. [BDW16] we need to revisit the original Misra-Gries algorithm briefly. Misra-Gires algorithm begins by setting up a table T containing k numbers of (v, c) pairs, with each pair being initialized to $(\perp, 0)$. $a_i \in \{1, 2, ..., n\} \cup \perp$ represents an item in the stream, while $c \in \mathbb{N}_0$ is a counter for v. When a new item a_i is inserted:

1. If $\exists (v, c) \in T$ s.t $v = a_i$: replace (v, c) by $(v, c + 1)$
2. Else if $\exists (v, c) \in T$ s.t $(v = \perp) \vee (c = 0)$: replace (v, c) by $(a_i, 1)$
3. Else: $\forall (v, c) \in T$, replace (v, c) by $(v, c - 1)$

The original paper by Misra and Gries [MG82], as well as the algorithm covered in our homework, primarily concentrated on obtaining k-frequent items without examining the precision of the related counts. It is later shown by Bose et al. [Bos+03] that when the algorithm is executed with $k = 1/\epsilon$, we have guarantees that, upon completion, the count c to each (v, c) pair is no more than $\epsilon \cdot m$ below its actual value, i.e. $f_i \geq c \geq f_i - \epsilon m$. This conclusion enables us to solve l_1-(ϵ, ϕ)-Heavy Hitters problem as provides the guarantee that, when $k = 1/\epsilon$ and $\phi \geq 2\epsilon$, if $f_i \leq (\phi - \epsilon)m$ then $i \notin S$.

A limitation coming from letting $k = 1/\epsilon$ is now Misra-Gires' space complexity becomes insensitive to the value of ϕ (it requires $O(\epsilon^{-1} \log n)$ if $k = 1/\epsilon$). To counter this, Arnab et al. proposed an algorithm that requires $O(\phi^{-1} \log n) + \epsilon^{-1} \log \epsilon^{-1})$ space, which can potentially save a significant amount of space compared to Misra-Gires when, for instance, $\phi \gg \epsilon$ or when ϕ is constant while ϵ varies with m. An example would be letting $\epsilon = 1/\log n$ and $\phi = 0.2$, the original Misra-Gires will give a space bound of $O(\log^2 n)$ while approach by Arnab et al. will have a better space bound of $O(\log n \log \log n)$.

2.2 Algorithm

Arnab et al. [BDW16] proposed the improved algorithm as follows:

Algorithm 1 **Improved l_1-Heavy Hitter**

```
1: Initialize:
2: $l \leftarrow 6 \log(6/\delta)/\epsilon^2$
3: Draw hash function $h$ from universal family $\mathcal{H} \subseteq \{h : [n] \to \lceil 4l^2/\delta \rceil\}$
4: An empty table $T_1$ of (key, value) pairs of length $\epsilon^{-1}$. Each key entry of $T_1$ can store an integer
   in $[0, \lceil 400l^2/\delta \rceil]$, and each value entry can store an integer in $[0, 11l]$. Note that table $T_1$ will be
   in sorted order by value throughout.
5: An empty table $T_2$ of length $\phi^{-1}$. Each entry of $T_2$ can store an integer in $[0, n]$. Note that each
   entry of $T_2$ corresponds to the id of the keys in $T_1$ for the highest $\phi^{-1}$ values.
6: procedure INSERT(x)
7: With probability $p = 6l/m$, continue, otherwise return
8: Misra-Gries Update$(x, h, T_1)$
9: if The value of $h(x)$ is among the highest $1/\phi$ valued items in $T_1$ then
10:    if if $x$ is not in $T2$ then then
11:       if $T_2$ currently contains $1/\phi$ many items then
12:          For $y$ in $T_2$ such that $h(y)$ is not among the highest $1/\phi$ valued items in $T_1$, replace $y$
             with $x$
13:       else
14:          Put $x$ in $T_2$
15:       end if
16:       Ensure that elements in $T_2$ are ordered according to corresponding values in $T_1$.
17:    end if
18: end if
19: return items in $T_2$ and corresponding values in $T_1$ iff value $\geq m \cdot (\phi - \epsilon) \cdot (6 \cdot l/m)$
```

Overview This algorithm aims to save space by doing two things:

1. Uniformly sample a subset $\mathcal{S}$ of $O(\epsilon^{-2})$ items from the data stream at random, it is proven by Arnab et al. [BDW16] that this subset suffices to represent the entire stream in computing

Algorithm 2 **Misra-Gries Update**

```
1: Input: item $x$, hash function $h$, Table of (key,value) pairs $T_1$
2: if $h(x)$ is in $T_1$ then
3:     Increment its counter value by 1
4: else if Size of $T_1$ is less than $\epsilon^{-1}$ then
5:     Insert $h(x)$ to $T_1$ with counter value 1.
6: else
7:     For all pairs in $T_1$, decrement counters by 1, remove the element if counter value becomes 0
8: end if
9: return $T_1$
```

l_1-Heavy Hitters.

2. Hash the stream items such that 1) the hashed item contains fewer bits, and 2) the hashed values do not collide, with the help of the reduced size of subset $\mathcal{S}$.

The size reduction, as a result, comes from the bits saved in table T_1 as a result of hashing. Table T_2 contains ϕ^{-1} items with $\log n$ bits each. Table T_1 contains ϵ^{-1} items which contains hashed value, and each hashed value requires $O(\log(\frac{4l^2}{\delta})) = O(\log(\epsilon^{-4})) = O(\log(\epsilon^{-1}))$ bits to store. Thus the total a space complexity is given as $O(\phi^{-1}\log n + \epsilon^{-1}\log\epsilon^{-1})$.

2.3 Implementation

Datasets We focused on testing l_1-Heavy Hitter algorithms for networking applications. Three real-world datasets are used for this purpose, they are: (Note that bold text should contain clickable hyperlinks to the websites where datasets can be downloaded from)

1. **IP Network Traffic Flows** This dataset contains network traffics under normal operations. It contains $3,577,297$ points and Heavy-Hitters are 10.200.7.218(8%) and 10.200.7.217(7%). This dataset is medium-sized with low-ϕ Heavy Hitters.

2. **DDoS Attack Dataset** This dataset represents the situation under a DDoS attack. The set contains $12,794,627$ points, the Heavy-Hitters are 172.31.69.25 and 18.219.193.20 which frequencies are about 14%. This dataset is large-sized with medium-ϕ Heavy Hitters.

3. **DDoS Attack under CoAP Protocol** This dataset also represents the situation under a DDoS attack. It conatins $624,937$ points and Heavy-Hitters are 192.168.1.9(46%) and 192.168.1.12(22%). This dataset is small-sized with high-ϕ Heavy Hitters.

Choice of ϵ and δ: We recall that our algorithm requires drawing hash function h from family $\mathcal{H} \subseteq \{h : [n] \to \lceil 4l^2/\delta \rceil\}$, and the saving of spaces comes from the reduced size of the hashed value. Thus the hashed range $\lceil 4l^2/\delta \rceil$ must not be larger than the range of $[n]$ itself. Note that we are examining the algorithm's application in network traffic monitoring, and the vast majority of the network traffic is in IPv4 address, which requires 32 bits to store. Thus the range of $[n]$ is confined to 2^{32} and in order to achieve bit saving the hashed range $\lceil 4l^2/\delta \rceil$ should be at most 2^{31} bits. Since we let $\epsilon = O(1/l) = O(\log n)$ the larger the value of epsilon the more bits we can save. On the other hand, ϵ value controls the lower bound of our ϕ-Heavy Hitter estimate, thus the larger ϵ is the worse our estimate will be. This creates a need for balancing between the number of bits saved and the quality of our estimated frequency. Also note that by increasing δ we can improve bit saving and quality of estimated frequency, at the expense of increased algorithm failure rate. See calculations and result below:

$$\text{let } \epsilon = \frac{y}{\log n}(\text{To achieve lower bound of } O(\log n \log\log n)) \tag{1}$$

$$\frac{4l^2}{\delta} = \frac{4 \times (6 \cdot \log 6/\delta)^2}{\delta} \times \frac{1}{\epsilon^4} \leq 2^{32} \times 0.5^z \quad (\text{bits saved} = z) \tag{2}$$

	$\delta = 0.1$	$\delta = 0.2$	$\delta = 0.3$
Save 1 bit	$\epsilon \geq 0.0695$	$\epsilon \geq 0.0533$	$\epsilon \geq 0.0452$
Save 2 bit	$\epsilon \geq 0.0827$	$\epsilon \geq 0.0634$	$\boxed{\epsilon \geq 0.0537}$
Save 3 bit	$\epsilon \geq 0.0984$	$\epsilon \geq 0.0754$	$\epsilon \geq 0.0639$
Save 4 bit	$\epsilon \geq 0.1170$	$\epsilon \geq 0.0896$	$\epsilon \geq 0.0760$

Table 1: Balancing between ϵ, δ, and bits saving, circled is the value set chosen for testing

We noticed that even by letting $\delta = 0.3$ and saving one bit only for each hash we still have a minimum $\epsilon \geq 0.045$. Thus we can safely conclude that, for IPv4 traffic monitoring, for any $\phi \leq 4\%$ this algorithm should not work as intended. Ideally, we would like $2\epsilon \leq \phi \leq 1$ such that our estimate does not deviate too far from true frequencies, thus any $\phi \leq 9\%$ is not recommended using this algorithm. However, as shown below, test results using real-world datasets still suggest that finding 0.065-Heavy Hitter is still doable with fair estimations.

2.4 Results

Estimation Accuracy The final y and δ chosen are 1.8(i.e. $\epsilon = 0.05625$) and 0.3 respectively. This is primarily because I want to actually achieve some bits saving while being able to find out the heavy hitters in the IP database (in which the most HH is merely 8%). These parameters are used on all three databases. ϕ values are set to 6.5%, 10% and 20% for IP, DDoS, and CoAP DDoS respectively. The table below summarized the results:

IPv4 addr	$\widehat{f_i}$ by Misra-Gires	$\widehat{f_i}$ by Improved l_1-HH	f_i	$\phi - \epsilon$
IP	-	-	-	-
10.200.7.218	0.0349	0.0364	0.0826	0.0088
10.200.7.217	0.0271	0.0275	0.0749	0.0088
10.200.7.199	0.0149	0.0153	0.0628	0.0088
DDoS	-	-	-	
172.31.69.25	0.0977	0.0977	0.1381	0.0438
18.219.193.20	0.0963	0.0962	0.1368	0.0438
CoAP	-	-	-	
192.168.1.9	0.4641	0.4673	0.4641	0.1438
192.168.1.12	0.2209	0.2217	0.2209	0.1438
192.168.1.5	0.1969	0.1960	0.1969	0.1438

Table 2: Comparison of estimated frequency $\widehat{f_i}$ with true frequency f_i

We noticed that the $\widehat{f_i}$ estimated by improved l_1-HH is not very different from the value estimated by the original Misra-Gires algorithm. The $\widehat{f_i}$ from improved l_1 can be higher or lower than $\widehat{f_i}$

by Misra-Gries since the improved algorithm samples from a subset. We also notice that when datapoint size is small and data distribution is relatively concentrated (e.g. in CoAP dataset), both algorithm finds the frequency that is close to the true frequency, especially Misra-Gires. However, when data size is big and more distinct points come across, the estimated f_i by both algorithms go closer to the lower $\phi - \epsilon$ bound.

Space and Time Complexity The time and space used for Misra-Gires(MG) and Improved $l_1(l_1)$ algorithms are summarized in table below: (Note that $p = 6l/m$ is the probability of Improved 1_1 algorithm to sample an item from stream)

Item	IP(MG)	IP(l_1)	DDoS(MG)	DDoS(l_1)	CoAP(MG)	CoAP(l_1)
Time(s)	33.9	30.7	178.7	112.8	2.88	2.89
Space of T_1	2584	2456	2200	2264	1712	1440
Space of T_2	-	696	-	456	-	256
Space of h	-	928	-	928	-	928
$p = 6l/m$	-	1.4%	-	0.38%	-	10.8%

Table 3: Space and time comparison, spaces are all in bytes

Space We can see that improved l_1 only decreases(if it does at all) a small amount of space in T_1 at the expense of adding another table T_2 and hash function h. Further testing suggests that, under IPv4 settings, the space saved in T_1 is not enough to compensate for the space added in T_2 and h under any reasonable ϵ and δ. To further reduce space used in T_1 we need either to lower the value of ϵ (Increase size of T_1) or reduce the size of the hashed range. However, we have shown previously that improving one meaning worsens another, and we can not improve both simultaneously without decreasing the success chance of the algorithm.

Time It has surprised me that the improved l_1 can actually run faster than the Misra-Gires algorithm at larger datasets. This probably is because the improved l_1 only needs to deal with a subset of the entire stream, even though it has to sort the T_1 and T_2 table every time when an item is inserted.

Beyond IPv4 We noticed that the value of the hashed range $\mathcal{H} \subseteq \{h : [n] \to \lceil 4l^2/\delta \rceil\}$ has no direct dependency on the value of n (unless we explicitly fix $\epsilon = y/\log n$) this means for fixed ϵ and fixed δ the spaced required for hashing is a fixed value. This is potentially very powerful in bits saving for large values of n. Notice that in Equation (2) if we change ϵ to a fixed value and solve for space required, calculations will show that we only need 41 bits of space to reach a ϵ value of 0.01. If this is used in IPv6 address (128 bit) then we can achieve an impressive space reduction of 87 bits per address and still have a very good estimate $\epsilon = 0.01$ on the true frequency of the addresses.

Unfortunately we were unable to find large enough datasets for IPv6 network traffic. Thus testing for IPv6 is done under custom-generated dataset (8 million points) which contains a 0.25-HH, a 0.15-HH, a 0.09-HH, a 0.07-HH, a 0.05-HH and others at random. We tested this dataset with $\phi = 0.08$ and $\epsilon = 0.007$, and we observed an reduction in space using improved l_1. We also noticed a significant increase in processing time for l_1. This can be improved by using e.g. quick sort in step 17 of Algorithm 1 instead of the bubble sort currently implemented. Results are shown in Table 4:

	l_1	MG	$\widehat{f}_i(l_1)$	$\widehat{f}_i(\text{MG})$	f_i	IPv6 addr (brief)
Time(s)	4284.9	138.1	0.24722	0.24719	0.25	627a:...:1263
Space of T_1	16296	18584	0.14715	0.14719	0.15	530b:...:1674
Space of T_2	784	-	0.08722	0.08719	0.09	f398:...:d4c9
Space of h	976	-	-	-	0.07	389e:...:f0c3
Total Space	$\boxed{18056}$	$\boxed{18584}$	-	-	0.05	a6f8:...:5cdd

Table 4: Comparison Misra-Gires and Improved l_1, all spaces are in bytes, note reduction in total space by Improved l_1(circled), also note that both $\widehat{f}_i$ agree up to 4 decimal places

3 Conclusion

Based on testing using real-world datasets. We concluded that the Improved l_1-Heavy Hitter algorithm by Arnab et al. [BDW16] can not achieve its goal of space reduction under the application of IPv4 network traffic monitoring. Nor can it perform approximating l_1 frequency for low ϕ or low ϵ under IPv4 settings. However, the improved processing speed of this algorithm by sampling a subset may be helpful in applications such as IPv4 traffic data processing. We also tested both algorithms under simulated IPv6 traffic. Preliminary results showed that the Improved l_1 Algorithm can achieve space reduction while maintaining accurate estimation under IPv6 settings. In our testing, the need of Improved l_1-HH for sorting both T_1 and T_2 becomes the limiting factor in the processing speed, which could impact on algorithm's performance (e.g in routers where processing time is limited) during heavy traffic or under low values of ϵ.

3.1 Future Works

Unknown m We noticed that our improved l_1-HH needs prior knowledge of the stream length m, which is impossible in applications such as monitoring traffic in routers. It has been shown by Arnab et al. that improved l_1-HH can be extended to unknown m length by running alternating instances of Improved l_1 Algorithm covering different intervals. More details can be found in their original paper [BDW16].

References

[BDW16] Arnab Bhattacharyya, Palash Dey, and David P. Woodruff. *An Optimal Algorithm for l1-Heavy Hitters in Insertion Streams and Related Problems.* 2016. arXiv: `1603.00213 [cs.DS]`.

[Bos+03] Prosenjit Bose et al. "Bounds for Frequency Estimation of Packet Streams." In: *Proceedings of the 10th International Colloquium on Structural Information and Communication Complexity.* Jan. 2003, pp. 33–42.

[Bra+15] Vladimir Braverman et al. *Beating CountSketch for Heavy Hitters in Insertion Streams.* 2015. arXiv: `1511.00661 [cs.DS]`.

[MG82] J. Misra and David Gries. "Finding repeated elements". In: *Science of Computer Programming* 2.2 (1982), pp. 143–152. ISSN: 0167-6423. DOI: `https://doi.org/10.1016/0167-6423(82)90012-0`. URL: `https://www.sciencedirect.com/science/article/pii/0167642382900120`.