Algorithms for Massive Data Problems

A B. Tech Project Report Submitted
in Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Devaishi Tiwari
(170101021)

under the guidance of

Dr. Benny George Kenkireth

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**Algorithms for Massive Data Problems**" is a bonafide work of **Devaishi Tiwari** (**Roll No. 170101021**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Dr. Benny George Kenkireth**

Assistant Professor,

Department of Computer Science & Engineering,

Indian Institute of Technology Guwahati, Assam.

Nov, 2020

Guwahati.

Acknowledgements

The topic "Algorithms for Massive Data Problems" was suggested by my guide Prof. Benny George Kenkireth to pursue for my thesis project. I am very grateful to all our faculties, friends and family who helped me on the way. I would like to thank my supervisor and guide Prof. Benny George Kenkireth for continuously helping and encouraging me.

Abstract

At this day and age, data is growing faster than ever. Around 300 billion emails, 65 billion messages and 350 millions pictures are sent out everyday. According to IDC's study – "The Digital Universe in 2020", we would have approximately 40 trillion gigabytes data by the end of the year 2020. This rapid growth in the amount of data has led to existence of a new field called **Big / Massive Data Analysis**

With this massive amount of data, comes a huge opportunity to increase **enhanced decision making, insight discovery**, and **process optimization**.

The objective of this project is to textbfintroduce the field of massive data analysis, communicate its usefulness in different sectors and examine some of the important algorithms related to massive data. In this report, we are going to discuss various deterministic and non-deterministic algorithms that deal with data that is too large to be stored in the Random Access Memory.

We will also look into various existing algorithms for handling and analysing a massive collection of data and evaluate their performance and correctness theoretically. Furthermore, we will try and implement some of these efficient algorithms and try to optimize the operations that are to be performed.

Contents

Chapter 1

Introduction

At this day and age, data is growing faster than ever. Around 300 billion emails, 65 billion messages and 350 millions pictures are sent out everyday. According to IDC's study – "The Digital Universe in 2020", we would have approximately 40 trillion gigabytes data by the end of the year 2020. This rapid growth in the amount of data has led to existence of a new field called **Big / Massive Data Analysis**.

According to Gartner, "Big Data are **high volume, high velocity**, or **high-variety** information assets that require new forms of processing to enable **enhanced decision making, insight discovery**, and **process optimization**." The concept of big data gained huge momentum in the 2000s. Even today, big data is one of the most significant fields of technology globally.

1.1 Motivation

This massive collection of data, called Big Data, can be effectively analysed for various useful insights. These insights can, in turn, enhance customer experience, reduce costs and increase efficiency. Some of the general benefits of Big Data Analysis are as follows:

- **Banking and Security**: Finance industry is highly dependent on data analysis for high frequency trading algorithms, stock analysis etc. Big Data can also be used to monitor the banks and trading markets and check for fraudulent practices.

- **Healthcare**: Healthcare industry generates a huge amount of data in the form of diagnostic reports and patients' records. This data can be analysed to detect symptoms of various diseases and help diagnose other patients.

- **Customer Behaviour Analysis**: The e-commerce industry has collected a lot of data over time, using customer loyalty cards, scanners and receipts. This data can then be used to analyze the customer's choices and behaviour and accordingly enhance the customer experience.

1.2 Objective

The objective of this project is to introduce the field of massive data analysis, communicate its usefulness in different sectors and examine some of the important algorithms related to massive data.

The main focus of the report is to **discuss various deterministic and non-deterministic algorithms that deal with data that is too large to be stored in the Random Access Memory.**

We will be looking into various existing algorithms for handling and analysing a massive collection of data and evaluate their performance and correctness theoretically. Furthermore, we will try and implement some of these efficient algorithms and try to optimize the operations that are to be performed.

1.3 Organization of the Chapters

This chapter provides a basic introduction on massive data analysis and its applications in the real world.

The remaining report is organized into chapters, as follows:

- Chapter 2 discusses three methods of computation used in massive data analysis.
- Chapter 3 describes frequency moment and data streaming methods in detail.
- Finally, Chapter 4 mentions sampling and sketching methods and explains some of the common algorithms for the same.

Chapter 2

Handling Massive Amounts of Data

There are various existing models of computation available to deal with massive data sets. The most efficient methods of computation are based on **data streaming, data sampling** and **data sketching**. These models will be discussed in detail in the subsequent chapters.

2.1 Data Streaming

In data streaming models, we assume that the data items arrive one at a time (sequentially), instead of being stored inside the memory as a block. We can better understand this concept by thinking of data items as packets on a computer network, passing across a router sequentially.

Let us assume that our data contains **n** data items, $\{a_1, a_2, \ldots a_n\}$ each of which has a size of **at most b** bits. Thus, our entire data would take **O(n*b)** space. But, in case of data streaming we only look into one data element (i.e. b bits) at a time which requires only **O(b)** space.

For example, let a_i ‘s be n integers that can have any value from 1 to $m = 2^b$. To find out the sum of elements in the data stream, we can just keep on adding the elements into a variable, say sum, stored in the memory. The sum of all the integers will be an integer from 1 to m*n, and thus can be stored in $log_2(m * n) = \mathbf{b + log_2 n}$ space.

This streaming model is useful for computing various statistics or operations over the data, like summation, average of items, or maximum, minimum, or majority items. We will discuss these algorithms in the next chapter.

2.2 Data Sampling

In the data sampling model, the input data is converted into its much smaller approximation and then stored in the memory. We can think of this as taking out a small set of people from a large population and using it for estimating various statistics or summaries over the entire data.

Data sampling is widely used to make standard matrix algorithms like matrix multiplication, single value decomposition, linear regression, etc faster.

However, there are certain class of problems, like counting the distinct elements, that cannot be solved using sampling methods. This limitation can be eliminated by using another method of computation, called data sketching.

2.3 Data Sketching

In the data sketching model, we go through the data stream and quickly process it to generate an in-memory summary for the whole collection of data. This compact summary/description that contains all the important information about the data items, is called a **sketch** of the data.

A great example of data sketching is web searches. Instead of caching the entire web pages, we only store a sketch of the web page in the memory. This sketch can then be used to find resemblance between the search string and the web page. We will discuss more about sketching and resemblances in the subsequent chapters.

Chapter 3

Working with Data Streams

In this section, we will discuss how specific operations like summation, average, minimum, most frequent etc. can be carried out over a massive data stream.

Let us assume that we have a data stream of length **n**, $\{a_1, a_2, a_3, ...a_n\}$, where a_i can take up any value from 1 to m. We will follow this notation throughout the chapter.

3.1 Frequency Moments

Frequency of a symbol i, denoted by f_i, can be defined as the number of times that symbol occurs in the data stream. Summation of the p^{th} power of these frequencies over all the symbols is called the $\mathbf{p^{th}}$ **frequency moment**.

$$F.M.(p) = \sum_{s=1}^{m} f_s^p$$

The frequency moment with p=0, is defined as the number of distinct symbols present in the stream. The first frequency moment will be the sum of frequencies of all the symbols in the stream, i.e.length of the stream itself. Second frequency moment, on the other hand, is somewhat related to the variance of the stream.

3.2 Algorithms using Data Streams

3.2.1 Count of Distinct Elements

Naive Method: The simple deterministic approach is to create a bit-vector in $\mathbf{O(m)}$ space to record which of the m symbols have occurred in the stream so far. Another method can be to store all the distinct elements in a set, which will take $\mathbf{O(n.log_2(m))}$.

Note that we need at least O(m) space for any deterministic algorithm to find out the count of distinct numbers.

Intuition: We can think of a non-deterministic algorithm that can estimate the value within a constant factor of the origin count, using randomization, with a small probability of failure.

Let **D** be the set of distinct elements present in the data stream. Let the minimum element present in D be **Min**. Now the expected value for min will be,

$$\begin{aligned} E(Min) &\approx \frac{m}{2}\ if\ |D| = 1 \\ &\approx \frac{m}{3}\ if\ |D| = 2\ and\ so\ on... \\ &\approx \frac{m}{|D|+1} \end{aligned}$$

Using the above equation, we can approximate the count of distinct elements as follows,

$$|D| \approx \frac{m}{Min} - 1$$

Algorithm: We can improve the accuracy of our algorithm by using a hash mapping over the set of symbols. We will use a hash function h where,

$$h : 1, 2, 3, ...m \longmapsto 0, 1, 2, ...M - 1$$

One example of a suitable hash function is as follows,

$$h_{ab}(x) = (ax + b)\ mod\ M$$

Here M is a prime number smaller than m.

We can store this hash function h_{ab} into memory, by storing the two coefficients a and b, which requires only $\mathbf{O(log_2M)}$ space. Now, we will store the minimum hash value, instead of the minimum element a_i. Thus, we can approximate the count using,

$$|D| \approx \frac{M}{Min}$$

Correctness: Our approximation of $|D|$ satisfies $\mathbf{\frac{d}{6} \leq \frac{M}{Min} \leq 6d}$ with a probability greater than $\mathbf{\frac{2}{3} - \frac{d}{M}}$.

3.2.2 Majority Element

A majority element is an element that occurs more than n/2 times in the stream. Let us consider the example of an election. Suppose there are **m** candidates and **n** number of votes, coming in as a stream of integers $\{\mathbf{a_1, a_2, a_3, ...a_n}\}$ where a_i can take up any value from 1 to m.

Naive Method: A simple deterministic approach can be to create a vector in $\mathbf{O(m.log_2(n))}$ space to record the number of occurrences of each symbol, in the stream. Another method can be to store all the distinct elements in a set, along with their frequency. This will take $\mathbf{O(n.log_2(n))}$ space.

Algorithm 1 Moore: Majority Element Algorithm

```
1: $element \leftarrow 0$
2: $count \leftarrow 0$
3: for $each\ i$ do
4:     if $i = element$ then
5:         $count \leftarrow count + 1$
6:     else
7:         $count \leftarrow count - 1$
8:     if $count = 0$ then
9:         $element \leftarrow i$
10:        $count \leftarrow 1$
```

Algorithm: We can maintain a single data element along with a counter. We initialise the element and the counter with 0.

For each incoming element a_i in the stream, we check if this a_i is equal to the stored element. If both the elements are equal, we increase the counter by one, else we decrease the counter by 1. The decrease counter step can be seen as cancellation of the current item and the saved item. At any point if the counter reaches 0, we replace the saved element with the current element and reset the counter to 1.

The saved value, at the end of the stream, is the majority element.

Correctness: The deterministic algorithm returns the correct majority element, provided there is a majority element in the stream. In case the stream doesn't have a majority element, the algorithm can return any random element. Thus, to ensure the correctness of the result, we can check whether the frequency of the returned element is greater than n/2 by going through the stream once more.

3.2.3 Most Frequent Elements

An updated version of the above majority element algorithm can be used to find out all the elements with frequency above a threshold. More specifically, we approximate the frequency (number of occurrences) of each element to within a constant difference from the exact value, i.e. $\mathbf{\frac{n}{k+1}}$, in $\mathbf{O(k.logn + k.logm)}$ space.

Algorithm 2 Misra-Gries: Frequent Elements Algorithm

```
1: $setElements \leftarrow \phi$
2: $count \leftarrow 0$
3: for $each\ i$ do
4:     if $a_i \in setElements$ then
5:         $count_i \leftarrow count_i + 1$
6:     else if $|setElements| < k + 1$ then
7:         $setElements \leftarrow setElements \cup \{a_i\}$
8:         $count_i \leftarrow 1$
9:     else
10:        for $all\ j \in setElements$ do
11:            $count_j \leftarrow count_j - 1$
12:            if $count_j = 0$ then
13:                $setElements \leftarrow setElements - \{j\}$
```

Algorithm: We can maintain a set of k data elements being counted along with their counters. Initially, the set is empty and the counters are set to 0.

For each incoming element a_i in the stream, we check if this a_i is equal to any of the stored elements. If it is equal to some element in the set, we increase the counter of that element by one. If the element is different from all the elements in the set, we decrement the counters for all the stored elements by 1.

If there are already k elements saved, we decrement the counters for all the stored elements by 1. If the counter for any element reaches 0, we delete that element from the set. The elements that are finally present in the set have frequency above $\frac{n}{k+1}$.

Correctness: For each element s in the stream, the frequency calculated by the algorithm, $\bar{f}_s$ is in the range,

$$\bar{f}_s \in \left\{ f_s - \frac{n}{k+1},\ f_s \right\}$$

where f_s is the actual frequency of the element. Note that the frequency of all the elements, which weren't saved in the set, is considered as 0.

3.3 Second Frequency Moment

The second frequency moment of a stream $\{a_1, a_2, a_3, ...a_n\}$ can be written as,

$$F.M.(2) = \sum_{s=1}^{m} f_s^2$$

3.3.1 Estimating Variance

The variance of the data stream can be written as the average of the variance of the frequency of individual symbols.

$$\begin{aligned} Var(X) &= \frac{1}{m}\sum_{s=1}^{m} Var(X_s) \\ &= \frac{1}{m}\sum_{s=1}^{m} \left(f_s - E(X_s)\right)^2 \end{aligned}$$

Since all of the symbols are equally likely to appear, the expectation of frequency for each symbol is, $\mathbf{E(X_s) = n/m \, \forall \, s}$. On substituting the value of E(X_s),

$$\begin{aligned} Var(X) &= \frac{1}{m}\sum_{s=1}^{m}\left(f_s - \frac{n}{m}\right)^2 \\ &= \frac{1}{m}\sum_{s=1}^{m}\left(f_s^2 - 2\frac{n}{m}f_s + \left(\frac{n}{m}\right)^2\right) \\ &= \left(\frac{1}{m}\sum_{s=1}^{m} f_s^2\right) - \frac{n^2}{m^2} \end{aligned}$$

3.3.2 Approximating Second Frequency Moment

Algorithm: To approximate the value of the second frequency moment, we define random variables $x_s \forall s, 1 \leq s \leq m$. These random variables can take a value of ± 1 with a probability $\frac{1}{2}$. Trivially, the expected value of each of these random variables will be 0 i.e.,

$$E\left(\sum_{s=1}^{m} x_s f_s\right) = 0$$

Now, the expected value of the square of the sum will be equal to,

$$\begin{aligned} E\left(\sum_{s=1}^{m} x_s f_s\right)^2 &= E\left(\sum_{s=1}^{m} x_s^2 f_s^2\right) + 2E\left(\sum_{s\neq 1} x_s x_t f_s f_t\right) \\ &= \sum_{s=1}^{m} f_s^2 \end{aligned}$$

Thus $\mathbf{a} = (\sum_{\mathbf{s=1}}^{\mathbf{m}} \mathbf{x_s^2 f_s^2})$ is a good estimator for the value of second frequency moment.

Correctness: To determine the accuracy of the algorithm, the calculate the second moment of a as follows,

$$
\begin{aligned}
E(a^2) &= E\left(\sum_{s=1}^{m} x_s f_s\right)^4 \\
&\le \binom{4}{2} E\left(\sum_{s=1}^{m}\sum_{t=s+1}^{m} x_s^2 x_t^2 f_s^2 f_t^2\right) + E\left(\sum_{s=1}^{m} x_s^4 f_s^4\right) \\
&= 6\sum_{s=1}^{m}\sum_{t=s+1}^{m} f_s^2 f_t^2 + \sum_{s=1}^{m} f_s^4 \\
&\le 3\left(\sum_{s=1}^{m} f_s^2\right)^2 = 3E^2(a)
\end{aligned}
$$

Therefore, $Var(a) = E(a^2) - E^2(a) \le 2E^2(a)$. Thus, by Chebyshev's inequality,

$$
Prob(|a - E(a)| > \varepsilon E(a)) = \frac{Var(a)}{\varepsilon^2 E^2(a)} \le \frac{2}{\varepsilon^2}
$$

Chapter 4

Sampling and Sketching

Data sampling is a method where we store only a small section of the data into the memory. This smaller section is used, instead of the original data to approximate computations in less space.

In **Data Sketching**, generally, the entire data is stored in a secondary memory. Unlike sampling, we read through all the data and create a small sketch/description which contains the important features of the original data and is stored in the primary memory.

4.1 Algorithms using Data Sampling

In this section, we will mainly discuss sampling in the context of matrix algorithms. Standard matrix algorithms like matrix multiplication, single value decomposition, linear regression etc. require around $\mathbf{O}(\mathbf{n^3})$ time for $n \times n$ matrices.

As the size of the matrix increases, these algorithms become slow. Hence, we need a faster method to estimate the results. In which case, we sample the rows and columns of the matrix to create a random sub-matrix to replace the original large matrix. We will discuss how to sample a matrix for estimating matrix multiplication in the next two sections.

4.1.1 Length Squared Sampling

To sample the matrix, we need to select a subset of rows or columns from the matrix. Let us assume that **s** number of columns are selected sequentially in s independent trials. Then, every column must be assigned a certain selection probability.

An intuitive approach could be to consider all the columns equally likely, i.e.**select the columns uniformly at random**. But this method wouldn't be efficient when most of the columns are close to the zero vector. This is because the algorithm will most likely select the insignificant columns, causing our matrix to be approximated as a zero matrix.

Thus, we will use **length squared sampling** method to select the columns instead. The squared length of a column c in a $n \times m$ matrix A is defined as,

$$S.L.(c) = \sum_{i=1}^{n} a_{ic}^2$$

In length squared sampling, we assign probabilities that are proportional to the squared lengths of the column. This way, significant columns will have more probability as compare to insignificant columns.

4.1.2 Matrix Multiplication

Let A be an $n \times m$ matrix and B be an $m \times p$ matrix. We need to approximate the product AB, which is,

$$AB = \sum_{k=1}^{m} A(:,k)B(k,:)$$

Here $A(:,k)$ denotes the k^{th} column of A, which is a $n \times 1$ matrix and $B(k,:)$ denote the k^{th} row of B, which is a is a $1 \times p$ matrix. Note that for each value of k, $A(:,k)B(k,:)$ are $n \times p$ matrices that add up to AB.

Algorithm: Suppose, we are selecting s number of columns, one at a time, in s independent trials. Let p_k be the probability of choosing the k^{th} column, based on length square sampling. Then, the matrix A can be sampled and scaled to a $n \times s$ matrix P with the columns as,

$$P = \left[\frac{A(:,k_1)}{\sqrt{sp_{k_1}}}, \frac{A(:,k_2)}{\sqrt{sp_{k_2}}}, \ \ldots \ \frac{A(:,k_s)}{\sqrt{sp_{k_s}}} \right]$$

And the matrix B can be sampled and scaled to a $s \times p$ matrix Q with the rows as,

$$Q = \left[\frac{B(k_1,:)}{\sqrt{sp_{k_1}}}, \frac{B(k_2,:)}{\sqrt{sp_{k_2}}}, \ \ldots \ \frac{B(k_s,:)}{\sqrt{sp_{k_s}}} \right]^T$$

Note that we have scaled P and Q to ensure that,

$$E(P.P^T) = E(A.A^T)$$
$$E(Q.Q^T) = E(B.B^T)$$

Thus, if we take **s** to be of O(1), the value of product PQ, which is an approximation of AB, can be calculated in $\mathbf{O(n^2)}$ **time**.

Accuracy: On substituting matrices A and B by matrices P and Q respectively, our matrix multiplication results in PQ such that,

$$E(\|AB - PQ\|^2) \ \leq \ \frac{\|A\|^2 \|B\|^2}{s}$$

4.2 Sketches of Documents

One of the important applications of data sketching is storing the context of web pages. Instead of caching millions of web pages, we can use sketching to create and store a brief sketch/description of the web pages in the memory.

This sketch can be used to estimate the similarity between two or more web pages. Before finding resemblances between web page sketches, we will discuss how to find resemblances between sets.

4.2.1 Resemblance in Sets

Let A and B be two sets of size 1000 each. We define the resemblance between two sets as,

$$resemblance(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

If we further increase the size of A and B, calculating the resemblance will become a tough task. One way to approximate the resemblance will be to randomly choose 10 elements from both A and B and check for the resemblance between them.

Note that even if the selected elements of A are present in B, they would not likely be present in the selected elements of B. Thus, the proposed method is not accurate.
The methods can be used to find out the value of resemblance more accurately are as follows,

1. Rename all elements using a random permutation and choose 10 smallest elements from both A and B for computing resemblance.

2. Select all the elements that are divisible by some randomly chosen integer m from both A and B. (element a_i is selected if a_i mod m = 0)

4.2.2 Resemblance in Web Pages

The method mentioned above can be extended over web pages, as the web pages can be considered as a string of words. We simply take the sequence of words in the page as a string, viewing each word as a character. This string can then be easily represented as a set of substrings of some length k.

Let S(W) be a set of substrings that are made up of k consecutive words from the web page W, then resemblance and containment can be defined as,

$$resemblance(A, B) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|}$$

$$containment(A, B) = \frac{|S(A) \cap S(B)|}{|S(A)|}$$

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The aim of this report was to **introduce the field of massive data analysis, understand its usefulness in different sectors and examine some of the important algorithms** related to massive data.

In this report, we explored various algorithms for massive data based problems. Along with the algorithm, we analysed the accuracy and failure probability of non-deterministic algorithms and established correctness for deterministic-algorithms. We divided these algorithms into 3 sections, based on the model of computation that was used.

In the data streaming model, we discussed algorithms to compute statistics on the data stream like counting distinct elements, finding out the majority element or most frequent elements. We also introduced the concept of frequency moments and discussed an algorithm to estimate the second frequency moment.

We also discussed sampling in context of the matrix based algorithms and examined an algorithm to approximate the matrix multiplication using sampling

Finally, we explored data sketching and examined how it can be used to estimate resemblance/similarity between two or more documents.

5.2 Future Works

The future work will include the following tasks,

- Replicating a large data stream to implement the streaming algorithms listed above.
- Implementing approximate matrix multiplication using sampling method.
- Implementing a sketching method to estimate the resemblance between two or more documents.
- Analysing the performance of these algorithms and comparing it with the theoretical results.

References

[BHK20] Avrim Blum, John Hopcroft, and Ravindran Kannan. *Foundations of Data Science*. Cambridge University Press, 2020.

[Cor16] Graham Cormode. *Misra-Gries Summaries*, pages 1334–1337. 01 2016.

[Cor17] Graham Cormode. *What is Data Sketching, and Why Should I Care?* 09 2017.

[Des07] Amit Deshpande. Sampling-based algorithms for dimension reduction. 09 2007.

[IW05] Piotr Indyk and David Woodruff. Optimal approximations of the frequency moments of data streams. pages 202–208, 01 2005.

[KNW10] Daniel Kane, Jelani Nelson, and David Woodruff. An optimal algorithm for the distinct elements problem. 01 2010.

[MZ10] Avner Magen and Anastasios Zouzias. Low rank matrix-valued chernoff bounds and approximate matrix multiplication. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, 05 2010.