University of Cincinnati

Date: 3/15/2011

I, Ravi Kumar Chatnahalli Sugandharaju, hereby submit this original work as part of the requirements for the degree of Master of Science in Computer Science.

It is entitled:
Gaussian Deconvolution and MapReduce Approach for Chipseq Analysis

Student's name: Ravi Kumar Chatnahalli Sugandharaju

This work and its defense approved by:

Committee chair: Raj Bhatnagar, PhD
Committee member: Anil Jegga, DVM, MRes
Committee member: Bruce Aronow, PhD
Committee member: George Purdy, PhD
Gaussian Deconvolution and MapReduce

Approach for Chipseq Analysis

A thesis submitted to the
Division of Research and Advanced Studies
Of
The University of Cincinnati
in partial fulfillment
of the requirements for the degree of

Master of Science

in the Department of Computer Science
Of
The School of Computing Sciences & Informatics
March 2011
By

Ravi Kumar Chatnahalli, Sugandharaju
BE in Computer Science
National Institute of Technology, Durgapur, India, 2005

Thesis Advisors: Dr. Raj Bhatnagar & Dr. Bruce Aronow
Abstract

Protein DNA interactions are one of the most fundamental factors used to determine how proteins and transcription factor binding sites affect the mechanisms related to a phenotype. The path breaking methodology CHIP-Seq which is a result of the combination of Chromatin Immunoprecipitation with parallel DNA sequencing in identifying the above interactions are of great interest. A methodology to identify the underlying peaks which are a direct representation of the protein interactions with the DNA is of paramount importance in identifying genes and pathways. We propose GaussDeconv, a novel Gaussian deconvolution approach, which is very precise in identifying these peaks, separating closely spaced peaks and it is also one of the first steps towards the comparative analysis of CHIP-Seq drawn from many experiments. We also propose a scaling mechanism for our computationally intensive algorithm, by successfully mapping the problem to a functional programming domain and applying Hadoop MapReduce to process the data in a distributed setup. We also show that the algorithm scales linearly with the increase of nodes in the cluster.
Acknowledgements

I would like to thank my advisors Dr. Bhatnagar & Dr. Bruce Aronow for their relentless guidance and their inspiration to work in such an interdisciplinary subject. I would like to thank Dr. Curtis Jamison under whom I started this project and all my colleagues at Cincinnati Children’s Research Foundation.

I would like to thank Dr. George Purdy for agreeing to be on my thesis defense committee.

I would like to thank my parents for their unceasing love and support throughout my graduate student life. I would also like to take this opportunity to thank my dear friends Amitabh Mishra, Kim, Victor, Chao Wu and Vineet for their knowledge and the several brainstorming sessions with them, which helped me to come up with interesting solutions to solve a particular problem.

Finally, I want to express my sincere gratitude to Cincinnati Children’s Bioinformatics Research Foundation for supporting me for most of my graduate studies.

Ravi Kumar Chatnahalli, Sugandharaju
# Table of Contents

Abstract .................................................................................................................................................. ii

Acknowledgements ................................................................................................................................. iv

Table of Contents ................................................................................................................................... v

List of Figures ........................................................................................................................................ x

List of Tables .......................................................................................................................................... xiii

Chapter 1: Introduction .......................................................................................................................... 1

1.1. Introduction ....................................................................................................................................... 1

1.2. Biology Details .................................................................................................................................. 1

1.2.1. Human Genome .......................................................................................................................... 2

1.2.2. Transcription Factor Binding Sites: ......................................................................................... 2

1.2.3. Genes ........................................................................................................................................... 2

1.2.4. Central Dogma of Molecular Biology ....................................................................................... 3

1.2.5. DNA Sequencing ...................................................................................................................... 4

1.2.6. ChIP Sequencing ...................................................................................................................... 4

1.3. Objective ......................................................................................................................................... 7
1.4. Organization of Thesis. ................................................................. 8

Chapter 2: Relevant Research ................................................................................................................................. 9

2.1. Major Steps in ChIP-Seq Peak finding Algorithms .................................................. 10

2.1.1. Preprocessing of Reads ................................................................................. 10

2.1.2. Signal Profiling ............................................................................................. 11

2.1.3. Peak Identification ........................................................................................ 11

2.1.4. Refinement of Peak result set ....................................................................... 12

2.1.5. Peak Prioritization ......................................................................................... 13

2.2. Algorithm Performance ....................................................................................... 13

Chapter 3: Methodology and Algorithm .................................................................................. 14

3.1. Overview ............................................................................................................. 14

3.2. Preprocessing of the Reads ................................................................................. 16

3.3. Signal Profiling ................................................................................................... 16

3.3.1. BinCount Algorithm ..................................................................................... 17

3.3.2. Signal Profile Generation Algorithm ............................................................ 22

3.4. Peak Identification and Splitting ......................................................................... 24
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.1.</td>
<td>Basics of Nonlinear Curve Fitting</td>
<td>25</td>
</tr>
<tr>
<td>3.4.2.</td>
<td>Nedler Mead Algorithm</td>
<td>25</td>
</tr>
<tr>
<td>3.4.3.</td>
<td>Gaussian Deconvolution Algorithm</td>
<td>31</td>
</tr>
<tr>
<td>4.1.</td>
<td>Introduction</td>
<td>37</td>
</tr>
<tr>
<td>4.2.</td>
<td>Drawbacks of traditional Distributed Parallel Computing</td>
<td>38</td>
</tr>
<tr>
<td>4.3.</td>
<td>Functional Programming Concepts</td>
<td>40</td>
</tr>
<tr>
<td>4.3.1.</td>
<td>Properties of Functional Programming</td>
<td>40</td>
</tr>
<tr>
<td>4.3.2.</td>
<td>Map and Fold Functions</td>
<td>41</td>
</tr>
<tr>
<td>4.3.2.1</td>
<td>Map Function</td>
<td>41</td>
</tr>
<tr>
<td>4.3.2.2</td>
<td>Fold Function</td>
<td>42</td>
</tr>
<tr>
<td>4.3.3.</td>
<td>Functional Programming and inherent Parallelism</td>
<td>43</td>
</tr>
<tr>
<td>4.4.</td>
<td>MapReduce Basics &amp; Architecture</td>
<td>43</td>
</tr>
<tr>
<td>4.4.1.</td>
<td>Map &amp; Reduce Interfaces of MapReduce</td>
<td>44</td>
</tr>
<tr>
<td>4.4.2.</td>
<td>Simple MapReduce Example</td>
<td>44</td>
</tr>
<tr>
<td>4.4.3.</td>
<td>MapReduce Architecture</td>
<td>45</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>4.5. Distributed File System</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>4.6. MapReduce Algorithm for ChIPSeq Deconvolution</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>4.6.1. Functional Programming Formulation</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>4.6.2. Main Algorithm of ChIPSeq MapReduce</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>4.6.3. Conclusion</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>Chapter 5: Results</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>5.1. BinCount Algorithm Results</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>5.2. Intersection of Deconvoluted Peaks with TFBS</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td>5.2.1. Results with 400bp window from Peak Center</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>5.2.2. Results with 200bp window from Peak Center</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>5.2.3. Results with 100bp window from Peak Center</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>5.3. Overall Peak Prediction</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>5.4. Comparison with MACS</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>5.5. MapReduce Results</td>
<td>71</td>
<td></td>
</tr>
<tr>
<td>Chapter 6: Conclusions &amp; Future Work</td>
<td>74</td>
<td></td>
</tr>
<tr>
<td>6.1. Introduction</td>
<td>74</td>
<td></td>
</tr>
</tbody>
</table>

viii
6.2. Conclusion .................................................................................................................. 74

6.3. Future Work .............................................................................................................. 75

References .......................................................................................................................... 76
List of Figures

Figure 1: Cross section of a DNA ................................................................. 2

Figure 2: The Central Dogma of Molecular biology .................................. 4

Figure 3: Detailed Workflow of ChIP Sequencing ...................................... 5

Figure 4: P53 Sequence Reads aligned to Human Genome ...................... 6

Figure 5: P53 enriched regions forming Peaks ......................................... 6

Figure 6: ChIPSeq Analysis Pipeline ......................................................... 14

Figure 7: Reads aligned to reference Genome ....................................... 16

Figure 8: Explanation of BinCount Algorithm with Sample Data. .......... 18

Figure 9: BinCount Output [ with binWidth = 200 ] .............................. 21

Figure 10: Signal Profile Algorithm & Result ......................................... 23

Figure 11: Nedler Mead - Reflection Point ............................................. 28

Figure 12: Nedler Mead - Expansion Point .......................................... 28

Figure 13: Nedler Mead - Contraction Point ........................................... 29

Figure 14: Nedler Mead - Shrink Point .................................................. 30
Figure 15: Gaussian Deconvolution Algorithm on a Sample ChIPSeq Dataset............. 35

Figure 16: Map function of Functional Programming.................................................. 42

Figure 17: Fold function of Functional Programming.................................................. 43

Figure 18: Life Cycle of MapReduce Workflow [11]..................................................... 46

Figure 19: Structure of Distributed File System [10].................................................... 49

Figure 20: ChIPSeq MapReduce Workflow.................................................................. 52

Figure 21: BinCount Algorithm Simulation with different sized bins. ....................... 58

Figure 22: Intersection of TFBS and Peaks with 400bp window................................. 60

Figure 23: Intersection of TFBS and Peaks with 200bp window................................. 61

Figure 24: Intersection of TFBS and Peaks with 100bp window................................. 63

Figure 25: Peak Height Prediction Histogram............................................................. 64

Figure 26: Comprehensive analysis of Peak Height Histogram method....................... 65

Figure 27: Peaks from ChIP Gaussian Deconvolution Algorithm.............................. 66

Figure 28: Peaks from MACS Algorithm..................................................................... 67

Figure 29: Juxtaposition of peaks from Gaussian Deconvolution & MACS............... 69

Figure 30: Detailed Analysis of ChipGaussDecon & MACS with Peak Ranges.......... 70
Figure 31: MapReduce Simulations on whole genome ChIPSeq Dataset ...................... 72
List of Tables

Table 1: Results with 400bp window from the Peak Center............................................ 59

Table 2: Results with 200bp window from the Peak Center............................................ 61

Table 3: Results with 100bp window from the Peak Center............................................ 62

Table 4: Analysis between Gaussian Deconvolution & MACS ...................................... 69
Chapter 1: Introduction

1.1. Introduction

The speed with which the DNA sequencing can be done these days has led to availability of large genome datasets and the improvement in the computer hardware has made it possible to analyze the same, much faster than before. Some of the most famous high throughput sequencing techniques in the industry currently are Massively parallel signature sequencing, pyrosequencing and Illumina sequencing.

CHIP-Sequencing is a technique which blends chromatin immunoprecipitation with high throughput parallel sequencing to identify the Protein DNA interaction sites [1]. The technique is also used to predict the transcription factor binding sites by interacting it with a protein of our choice. The protein interaction with DNA in the regulation of gene expression is very important in the complete understanding of Phenotypes, biological process, diseases and the resulting research on drugs for diseases.

1.2. Biology Details

The section explains in brief the biological fundamental and terms upon which the forthcoming chapters are built.
1.2.1. Human Genome

The human genome constitutes a sequence of approximately 3 billion component parts, called nucleotides, which are organized into DNA molecules - the double helix. The nucleotides are comprised of four symbols: A, C, G, and T, corresponding to adenine, cytosine, guanine, and thymine. These nucleotide symbols codes for the sequence of amino acids the body will use to build proteins.

*Figure 1: Cross section of a DNA*

1.2.2. Transcription Factor Binding Sites:

Transcription factors are the authoritative proteins which bind to specific DNA sequences and control the transfer of genetic information from DNA to mRNA. They are definitive in the activation or suppression of RNA polymerase to specific genes.

1.2.3. Genes

Genes are functional subunits of DNA. It resides on certain coordinates of DNA and RNA that codes for a specific type of protein or for an RNA chain that has a function in the
organism. Each gene contains a particular set of rules also referred to as Genetic code, usually coding for a particular functional protein.

### 1.2.4. Central Dogma of Molecular Biology

Central Dogma of Molecular biology is the fundamental framework of Molecular Biology which forms the basis in understanding the transfer of genetic information in living organisms and this process take place in 4 stages.

- **DNA Replication**: DNA replicates and encodes information in this process.

- **DNA Transcription**: DNA synthesizes mRNA and during this process the information is coded into the newly produced mRNA with the help of RNA polymerase and transcription factors.

- **mRNA Movement**: mRNA is processed and it moves from the nucleus to cytoplasm.

- **mRNA Translation**: Ribosomes read the information from the mRNA and uses it for Protein synthesis.
Figure 2: The Central Dogma of Molecular biology

Figure 2 gives a graphical representation for the above mentioned 4 steps of the framework.

1.2.5. DNA Sequencing

Wikipedia refers to DNA Sequencing as “methods for determining the order of the nucleotide bases—adenine, guanine, cytosine, and thymine - in a molecule of DNA”. Biological research and breakthroughs have been expedited by the used of DNA sequencing. It has almost become a quintessential part of the research these days.

1.2.6. ChIP Sequencing

ChIP Sequencing(also known as ChIP-Seq), is a sequencing technique used to study the protein interactions with the DNA. It is a combination of two process which is chromatin Immunoprecipitation followed by high throughput parallel sequencing of the short reads.

The detailed explanation of the ChIP Sequencing method is explained below:
ChIP Sequencing Workflow contains two major processes. In Figure 3, Steps 1 to 5 represent the Chromatin Immunoprecipitation (ChIP).

Firstly, the DNA is enriched by a specific protein of interest, after the crosslink the DNA is sheared, treated with Protein specific antibody and immunoprecipitated. Secondly, DNA is reverse cross linked, purified and prepared for the next method which is Sequencing.

Step 6 in the above figure is called the High throughput Sequencing. After the first method, the whole DNA is now converted into fragments (also called reads). These reads now have to be aligned to the genome reference in order to identify the specific location from where it came. The parallel sequencing is done using many different methods, of which the most famous are Illumina Sequencing & Bowtie.

Figure 4 shows the aligned reads on the human reference genome Assembly hg18. The black rectangular bars represent the short reads. It also shows the direction arrow on them which represent the direction in which they align in the double helix structure.
Figure 4: P53 Sequence Reads aligned to Human Genome

When these reads, cluster near a particular region on reference genome, they are called “Enriched” regions. These overlapping regions of enrichment form peaks. These peaks also signal the presence of transcription factor binding sites and a gene nearby.

Figure 5: P53 enriched regions forming Peaks

Figure 5 shows the graphical representation of the peaks formed by these enriched regions.
1.3. Objective

The prime objective of the research was to build a pipeline which would help researchers to identify more precisely, the transcription factor binding sites from the ChIPSeq data. The current approaches resort to sub-optimal solutions in terms of exact binding site predictions while gaining on the speed. However, our research takes a novel approach with the central idea of Gaussian deconvolution towards the analysis of the ChIPSeq peaks which better localizes the binding sites than any other approach.

The pipeline developed contains clearly modularized steps towards the analysis of the data. One of the major concerns of the previous approaches was their inability to separate the closely spaced peaks which might represent two binding sites in close vicinity. Pepke et al. review paper on major ChIPSeq approaches do affirms the need for a Peak Deconvolution approach rather than the traditional peak identification approach assumed by most software’s for identifying the binding sites. The main reason the deconvolution approach was not an attractive choice was because of scalability of the algorithm for large genomes. Peak Deconvolution is a digital signal processing problem and is in no way an inexpensive algorithm. In this thesis, we address this particular concern by drawing inspiration from a completely new field of functional programming and distributed data processing. The latter part of the research successfully maps this intricate biological problem into a functional programming domain and applies MapReduce to get some interesting benchmarking.
1.4. Organization of Thesis.

The thesis is organized into 6 major chapters. Chapter 1 introduces the problem and explains the basics of biology and the specific objective of the research. Chapter 2 discusses thoroughly the related research in the analysis of ChIPSeq and the drawbacks of the existing approaches. Chapter 3 describes the methodology & algorithm. Chapter 4 introduces the concepts of Functional programming, MapReduce and discusses the algorithm developed to solve the scalability issue. Chapter 5 discusses the results obtained on the data sets. Finally, Chapter 6 concludes the paper with a brief discourse on future work.
Chapter 2: Relevant Research

This chapter surveys the relevant ChIP-Seq software’s that have been built in the past few years. We discuss some of their implementation details, algorithms, merits and demerits on which we build upon our thesis which addresses some of the concerns of the existing software’s.

In Chapter 1 we discussed in detail the workflow of ChIP-Seq. After the sequencing step, the millions of short reads are accurately mapped to the genome reference which becomes the primary input to all the software which mines for meaningful information like peak finding, transcription factor binding site detection from them.

Pepke et al. came up with a review paper which discussed in detail the steps involved in the analysis of ChIP-Seq data and thoroughly profiled about 12 different programs for detecting peaks from the ChIP-Seq data [5]. Recently, Wilbanks et al. discussed a paper wherein they evaluated the algorithmic performance of some of the standard ChIP-Seq Peak detection algorithms [6]. We will briefly summarize these 2 papers; discuss few other works which will give us a holistic picture of the current state of the research in the analysis of ChIPSeq.
2.1. Major Steps in ChIP-Seq Peak finding Algorithms

A close review of the existing peak finding algorithm reveals standard modules which forms the algorithm as a whole [5]. These modules are:

- Pre Processing of Reads
- Signal Profiling
- Peak Identification and Splitting
- Refinement of the Peak result set.
- Peak Prioritization.

2.1.1. Preprocessing of Reads

Preprocessing of Reads is the first step which helps us to eliminate the strand bias modality [2] which arises due to the inherent bias introduced by the ChIP-Seq reads. In the Watson and Crick model of DNA which is also called as the Double Helix, the direction of nucleotides in one strand 3’(3 Prime) is opposite to the other end 5’(five Prime). The reads are sequenced from either 5’ or 3’ end and they are expected to align same number of times in expectation and this creates two distributions, but they are related. Most of the software’s tend to shift tags in 3’ direction or extend the tags by a certain fragment size. Tag shifting is done by using the
value given by user input [3, 4] or by sampling a few high resolution peaks [2]. This step is however unavoidable and is part of every implementation and they are all very similar.

2.1.2. Signal Profiling

The overlapping reads referenced on the genome, when vertically cut at a genomic position gives us a read count (at that genomic position). These read counts when combined together give us a signal, which subjectively represents long running “Segments” on genome. These Segments typically are formed by few tens to hundreds of overlapping reads and it might have one or more than one peak. Signal Profiling is a way of identifying these large Segments which directly represents the enrichment regions on genome. This step is probably one of the most important steps which directly affect the peak finding algorithms which succeeds this step. These Segments are selected by either aggregation of overlapped reads [3, 4] or by tag shifting using a scan window [2]. We select these Segments using an algorithm called BinCount Algorithm.

2.1.3. Peak Identification

The Signal Profile (or Segments) and tag counts in that Segment is the input to the Peak Identification module which actually determines the Peaks in the Segments. The measure which determines the Good Segment from the Bad Segment varies from implementation to implementation. FindPeaks [3] uses Height threshold as the main criteria whereas MACS and
PeakSeq come up with local region $P$ value calculated from Poisson and binomial distribution\[2, 4\]. However, these are successful in determining enriched regions, they are not precise in identification of underlying peaks. It is possible to have many peaks overlapped due to closely spaced TFBS to form a Segment with good height and shape. In this case, they need to be separated which most of the peak identification algorithm fails. We propose a Gaussian Deconvolution approach to split these peaks from a given segment which addresses this issue and helps in precise identification of the binding sites.

2.1.4. Refinement of Peak result set

After the peak Identification step, every software applies some kind of a filter to remove artifacts from the peaks that are produced from the previous step \[5\]. The filter is applied to remove the noise from peaks which might be unduly influencing the shape, size of peaks and in turn influencing ranking of the peaks. FindPeaks does this by separating multiple peaks by using a minimum relative valley depth which is required between two adjacent peaks followed by Peak trimming to only maintain regions of interest \[3\]. MACS and PeakSeq both doesn’t apply a filtering to remove the artifacts \[2, 4\]. The main concern in this step is that the filters might remove regions which might be part of the peak and it is a hard problem to make this distinction. Pepke et al. agrees to this as an active area of research. We addressed this issue in our Gaussian Deconvolution Algorithm where the unwanted artifacts were removed by separating them out as small peaks with less height without invading into a good peaks region. This step is instrumental in removing the unwanted peaks from the final result set.
2.1.5. Peak Prioritization

Peak prioritization is the last module in the ChIP-Seq Analysis which uses several features of the peak to provide a scoring mechanism; this step is also referred as *Peak Ranking* [5]. FindPeaks [3] ranks its peaks by using the number of reads under the peak as criteria. MACS uses a local Poisson $P$ value which it calculates during the Peak Identification phase [2]. Similarly, PeakSeq uses a local binomial $Q$ value for the same. We use the Peak Height and Peak Area (number of reads under the peak) as a criterion for the Peak prioritization [4].

2.2. Algorithm Performance

Gaussian Deconvolution is the main module of our algorithm which does the peak identification and generates peak statistics. The algorithm is based on a iterative curve fitting approach and hence a very expensive approach. Our approach can scale to many of the smaller organisms and their relatively smaller genome. However, a human genome with 3 billion base pairs would take considerable amount of time. We address this main concern by drawing inspiration from a completely new field of functional programming and distributed data processing. The problem is mapped to a functional programming domain and then a MapReduce is applied and processed in a Hadoop cluster with several nodes to get good throughputs. We also show that, this approach linearly scales with the scaling of number of nodes in the cluster.
Chapter 3: Methodology and Algorithm

3.1. Overview

This Chapter describes the methodology and core algorithms of different modules of our approach in the Analysis of ChIPSeq. Analysis of ChIPSeq is an involved process with many modules and this requires each module be built as a pipeline where the output of one stage goes as input to the next module.

In Section 2.1, We discussed some of the standard modules used by the existing algorithms [2, 3, 4] and we conform to the same standard set of modules.

![ChIPSeq Analysis Pipeline](image)

*Figure 6: ChIPSeq Analysis Pipeline*
Figure 6 shows a high level overview of the ChIPSeq pipeline which we built and it involves the following steps. Step 1 involves preprocessing of the reads, which removes some of the biases of the sequencing techniques & duplicate reads. In Step 2, we take the reads which were processed from the previous step to generate enriched segments using BinCount Algorithm, construct Sample set and a Signal profile. Step 3 takes as input the Signal Profiles in the form of Segments and applies the Gaussian Deconvolution Algorithm to give peaks and statistics related to peaks. Each of these steps involves several parameters that go into these algorithms which are either learned from the dataset or provided by the user. Finally, Steps 4 & 5 are involved in the refining of the peaks and identifying the most important peaks.

The input to the ChIPSeq pipeline as we discussed is a file with reads already aligned to the standard reference genome. A typical alignment file contains millions of reads which includes all the Chromosomes. The sample of the same is shown in Figure 7 below. Each line represents a single read. It contains a lot of information about its position, associated chromosome, actual base pair sequence and some alignment related details. The most important columns are marked as rectangular boxes in different colors.
Figure 7: Reads aligned to reference Genome

The black, blue, red and green rectangular boxes represent the actual base pair sequence, read start position co-ordinate on genome, forward or reverse strand and length of the read respectively. These are some of the most important data which we shall use during the next few steps of our processing.

3.2. Preprocessing of the Reads

The preprocessing is an inevitable step which generates good resolution of Signals at a later stage. The procedure is same in most of the standard approaches. This step involves removing of the duplicate reads at a specific genomic location and shifting of the reads towards the three prime end. There is a bimodal enrichment pattern observed due to the Watson and Crick tag enrichment which we take advantage, in finding the enrichment region [2].

3.3. Signal Profiling

After the preprocessing of the reads, the identification of the enriched regions on genome and building a signal profile is the most important step. “Enriched” region means the areas where the reads are clustered around a particular site. There are millions of reads and they get mapped to positions on genome with different density depending on the quality of the binding site. So, numbers of overlapping reads are directly proportional to the quality of a transcription factor
binding site. However, the different densities with which the reads get mapped to genome leave us with long regions of genome which are sparse. The removal of these sparse regions and retaining the regions of interest is in itself a challenging problem. Intuitively, increasing the baseline is one of the solutions since the increase of baseline affects every region across genome. But, this results in loss of information in the basal regions.

This particular step involves an algorithm called BinCount algorithm which helps in identifying enriched regions followed by a Signal Profile building algorithm which produces Segment signals which will finally go as an input to the Deconvolution algorithm.

### 3.3.1. BinCount Algorithm

BinCount Algorithm is a standard algorithm used widely in the Bioinformatics domain. The basic idea behind Algorithm is to divide the whole genome rather each chromosomal regions into bins of width \textit{binWidth}. Now, Scan through the reads in the sorted order to assign each read to a particular bin.

A read is assigned to a particular bin only if it is completely within that bin and for the ones in the boundary; tie is broken by assigning it to the bin which has a maximum overlap.
Figure 8: Explanation of BinCount Algorithm with Sample Data.

To explain the above idea in detail, Consider Figure 8 which shows a sample dataset where the blue bar at the below represents the chromosome region and it has been divided into 6 different bins (labeled as A, B, C, D, E and F) with binWidth=200. The small green bars represent reads and they are located at specific chromosomal locations. So, according to the algorithm we assign each read to a specific bin and get a count for each bin.

Once all the reads have been assigned to their specific bin, we have a Bin Count for each bin,

\[
\begin{align*}
    binCount(A) &= 2, \\
    binCount(B) &= 3, \\
    binCount(C) &= 4, \\
    binCount(D) &= 6, \\
    binCount(E) &= 5, \\
    binCount(F) &= 2.
\end{align*}
\]
After this step, a BinCount threshold is chosen and all the bins below that threshold are eliminated from further consideration. For instance, in the above example, if we choose BinCount threshold as 2, the bins A and F are removed and will not be used when we build the Signal profile or Segment. On the other hand, the bins B, C, D, and E qualify to form a Segment as they all have a count above the chosen threshold. These qualified contiguous bins are concatenated and they form our regions of Enrichment.

**BinCount Algorithm**

**Input:** ChIPSeq Alignment File

**Output:** Enriched regions [or Bins]

**Step1:** Initialize the alignment file for Reading.

**Step2:** Initialize the binWidth

**Step3:** Extract the next read and gather chromosome, readStartCord & readLength.

**Step4:** For each Chromosome chromosome, created a HashTable hashTable [ If not already created.]

**Step5:** Generate hashKey = floor ( readStartCord / binWidth )

**Step6:** Set BinCount = hashTable.getValue( hashKey );

**Step7:** BinCount = BinCount + 1;

**Step8:** Do hashTable.insert ( hashKey , BinCount );

**Step9:** If all reads of chromosome chromosome completed, Go to Step 3 else Step 10.
Step 10: Sort the hash table in ascending order of their hash keys and persist the bins into a file as

\[
\begin{array}{ccc}
\text{chromosome} & \text{hashKey(Bin Start Co-ordinate)} & \text{hashValue(Bin Count)} \\
\end{array}
\]

Step 11: End

In the above algorithm, the input is the sequence alignment file which contains reads aligned to a particular genome reference and \textit{binWidth} which is the size of the bins. The best bin size to choose will be discussed later in the section. Step 1 & 2, Initializes the sequence file to read and the \textit{binWidth}. Step 3 starts reading one read at a time and gathers the \textit{chromosome}, \textit{readStartCord} & \textit{readLength}. Step 4 creates a Hash Table, if there was not already one that was created for the \textit{chromosome}. Step 5 creates \textit{hashKey} and it is the floor of \textit{readStartCord} / \textit{binWidth}, this gives us the start position of a particular bin and that will be our hashKey. Step 6 & 7 calculates the value \textit{BinCount} value by extracting the existing value and by incrementing it. Step 8 updates back to Hash Table with \textit{hashKey} as it’s Hash Key and \textit{BinCount} as it’s Hash Value. Step 9 loops back to fetch the next aligned Read. Finally, Step 10 sorts the values in the ascending order and outputs all the bins and their respective counts.

BinCount Algorithm gives us the bins across chromosome and their counts. These counts are definitive in the prediction of the \textit{Enriched} regions. Choosing a reasonable \textit{binWidth} is very crucial in the selection of \textit{Enriched} regions. Decreasing the \textit{binWidth} and asking for bins above a certain \textit{threshold} might be too restrictive and could result in a pruning, large part of genome. For Instance, if the \textit{binWidth} is 100 and \textit{threshold} is 5 then this condition restricts
more regions when compared to $\text{binWidth}$ is 200 and $\text{threshold}$ is 5. The probability of reads occurring in binWidth=200 is more than the binWidth=100.

**Figure 9: BinCount Output [with binWidth = 200]**

In Figure 9, We have plotted the variation of number of Reads/bin on $X$-axis and their corresponding log of number of bins on $Y$-axis, for a ChIPSeq dataset. The algorithm was run with a $\text{binWidth}$ of 200. We clearly see that, In chromosomal regions, the bins with BinCount 1, 2, 3 are some of the highest and the logarithm of number of bins is decreasing with the increase in number of Reads/bin. The red line in the above figure shows the threshold which was chosen, is the pruning criteria for the bins and it is called BinCount Threshold. For Instance, If the BinCount Threshold was chosen to be 5, that means we reject all the bins with
binCount < 5. The argument is that, the possibility of a peak formation out of 5 reads within a binWidth of 200 and its biological significance is highly improbable and it is better we remove such bins from any further analysis.

The Algorithm effectively removes all the sparse regions of chromosome and leaves us with enriched regions of interest. The next step is to find the Signal from these selected bins which will then be used for Peak Deconvolution.

### 3.3.2. Signal Profile Generation Algorithm

In this section, we shall build a Signal Profile from the qualified bins from the BinCount Algorithm. The high level overview of the algorithm is that the contiguous bins are concatenated to form a segment and then for each genomic location in that region, a count of number of overlapping reads are calculated and this would give a signal and this is what is called the Signal Profile.
Figure 10: Signal Profile Algorithm & Result

To understand the algorithm in detail, Consider Figure 10, this diagrammatically explains the algorithm. In Part I of the figure, It shows 3 bins A, B, C with binWidth=5 which was obtained from the BinCount Algorithm. The green bars represent the reads that are aligned to the genomic co-ordinates between 1 and 16. The red dotted lines are drawn to get the count of number of overlapping reads which gives the numbers at the top which are called Read Counts.
In *part II*, these Read Counts are joined together to get the *Signal Profile*. This Signal Profile is fed as input to our next step in pipeline, which is Peak Deconvolution.

### 3.4. Peak Identification and Splitting

This section describes the module of Peak Identification and splitting, which are both an important aspect of finding the best peaks amongst the millions of peaks across a genome. Pepke *et al.* summary paper in their closing remarks do stress on the fact that, the analysis of ChIPSeq is more of a Peak Deconvolution problem than just a Peak Identification problem [5]. As we discussed in Section 2.1.3, most of the existing implementations don’t try to split the peaks and there are a few which do split the peaks, but they are not very precise in their approach.

The reason for most implementations not taking the Deconvolution approach is due to the computationally expensive nature of the solution. But, they all suffer from the precise prediction of significant peaks and the underlying closely spaced transcription factor binding sites (TFBS). Our Algorithm takes the novel approach of Gaussian Deconvolution and addresses the scalability of the solution with the MapReduce approach.
This module takes as input the Signal Profile generated in the previous section which we call *Segments* and applies the Gaussian Deconvolution to split all the underlying peaks within. The assumption is that the DNA fragments binding to the locations on genome follows a Gaussian distribution. The algorithm is based on the downhill simplex method or Nedler Mead technique developed by Nedler and Mead. This is an iterative method which is used for minimization of an Objective function in a $N$-dimensional space [8].

### 3.4.1. Basics of Nonlinear Curve Fitting

The main aim of curve fitting is to come up with a Mathematical model that best extrapolates the data. Some of the well-known linear curve fitting algorithms are linear least squares and polynomial least squares. The main idea of these algorithms is the find the co-efficient of the mathematical function which gives the shape for the curve. However, the major limitations of these methods are when the dependent variable has nonlinear relationship with the function values. So, the best way to fit these nonlinear relationships is by trial and error method, which is usually referred to as Iterative Curve Fitting [9].

### 3.4.2. Nedler Mead Algorithm

Nedler Mead algorithm is a simplex method used to find minimum of many variables and we use it as a core mathematical formulation for our Iterative curve fitting algorithm [8]. Wikipedia describes simplex as, “$N$-simplex is a $N$-dimensional polytope which is a convex
hull of its \( N+1 \) vertices”. For Instance, If \( N=2 \), The 2-simplex is a triangle. First, Let us elaborate the Nedler Mead algorithm and get the intuition behind the algorithm and then discuss its usage in Curve fitting.

Consider 2 variables, as we said before Simplex is a triangle. The high level idea of Nedler Mead algorithm is, the algorithm starts by creating a triangle from the initial guesses of the variables. It calculates the function values at the vertices of the triangle [8]. The vertex with the largest function value is rejected for a new vertex where the function value is less, which forms a new triangle with the other two points. The algorithm iteratively builds many triangles and there would be a point where triangle shrinks to a point. This is the point which we are interested in and we call it the local minima.

Let us consider, function \( f( x , y ) \) whose minima needs to be found. To initiate the Nedler Mead, we have 3 vertices of a triangle.

\[
S_i = ( x_i , y_i ), \ i = 1, 2, 3.
\]

The function values at these 3 points are: \( V_i = f( x_i , y_i ), \ i = 1, 2, 3 \). Then, \( V_1, V_2 \) and \( V_3 \) are sorted by their function values and arranged as \( V_1 \leq V_2 \leq V_3 \).

\[
G_1 = ( x_1 , y_1 ) \quad \text{ - First Best Point}
\]
\[
G_2 = ( x_2 , y_2 ) \quad \text{ - Second Best Point}
\]
\( G_3 = (x_3, y_3) \) - Third Best Point

We have 3 points, with \( G_1, G_2 \) & \( G_3 \) being the best points in order and they have increasing function values. The idea is to replace \( G_3 \) by some other point which is better and form a new triangle and continue the algorithm again till we hit the local minima.

**Step 1:** Calculation of Midpoint

This step calculates the midpoint of the line joining the First Best Point(\( G_1 \)) & Second Best Point(\( G_2 \)). Let the midpoint be \( G_M \).

**Step 2:** Calculation of Reflection Point

The intuition is that the function takes lesser value when moving from \( G_2 \) to \( G_1 \) and from \( G_3 \) to \( G_2 \). So, the next best point might be on the other side of the segment \( G_1G_2 \).
**Figure 11: Nedler Mead - Reflection Point**

*Figure 11* shows the construction for the Reflection Point GR. After the calculation of Midpoint $G_M$, extend the line from $G_3G_M$ by distance ‘$a$’[distance of $G_3G_M$] to get the point $G_R$. The reflection point calculation is an important step towards reaching to the next best point.

**Step 3: Calculation of Expansion Point**

If the function value at Reflection point $G_R$ is found to be less than $G_3$, it is considered a good progress. Now, We would like to expand a little further and check for an even better point.

![Diagram showing reflection and expansion points](image)

**Figure 12: Nedler Mead - Expansion Point**

In *Figure 12*, We expand from the point $G_R$ in the direction of vector $G_3G_M$ by a distance of ‘$a$’ and we get the point $G_E$, which is our Expansion point. If the function value at $G_E$ is less
than \( G_R \) then we have found a better point for forming the next triangle else we choose the Reflection Point \( G_R \).

**Step 4: Calculation of Contraction Point**

If the function value at \( G_R \) is same or worse than \( G_3 \), We need to check few other points in the vicinity. Let us consider two midpoints \( G_{C1} \) and \( G_{C2} \), which are the midpoints of \( G_3G_M \) and \( G_MG_R \) respectively.

![Nedler Mead - Contraction Point](image)

**Figure 13: Nedler Mead - Contraction Point**

As Figure 13 shows, the construction of \( G_{C1} \) and \( G_{C2} \) gives us two points to check for function values. If any midpoint is better than \( G_3 \) and better than the other midpoint, we
choose it as our next best point and let’s call it $G_C$. We have a new triangle with the vertices $G_1G_2G_C$ and we proceed towards our next iteration.

**Step 5: Calculation of Shrink Point**

In this step, we make an attempt to shrink towards the best point available till now $G_I$ if the function value at $G_C$ is not satisfactory.

![Figure 14: Nedler Mead - Shrink Point](image)

*Figure 14: Nedler Mead - Shrink Point*

In *Figure 14*, a new construction is made, the shrink point which is the midpoint of $G_I G_1$. Now, we have a new triangle $G_I G_S G_M$ which is constructed by excluding 2 points $G_3$ and $G_2$. The next iteration is continued from the newly constructed triangle.
To conclude, we make all these above mentioned decisions and continue to iterate through the sequence of triangles. The size of triangles continues to decrease as we move towards the local minima and finally it shrinks to a point. This is when the Nedler Mead algorithm terminates having found the minimum value of the function.

3.4.3. Gaussian Deconvolution Algorithm

Gaussian Deconvolution algorithm is one of the main algorithms of our approach which takes in Signal Profile / Segment generated in the previous step to produce individual Gaussian peaks. The idea is to split the peaks in such a way that they additively form the Signal Profile. Nedler Mead algorithm discussed in the previous section is at the core of this algorithm and which does the Gaussian function optimization.

The first step of this algorithm is to randomly select the Segments and thoroughly fit these Segments with increasing number of peaks and record the Root Mean Square Error (RMSFitError) generated between the Signal Profile and the newly fitted Gaussian peaks [9]. Tom O'Haver discusses different strategies of how to come up with a minimization function for curve fitting[9]. The maximum number of Peaks to be fitted can be derived from this random sampling by choosing a reasonable RMSFitError and the corresponding number of peaks fitted. This becomes our PeakFitThreshold.

---

**Gaussian Deconvolution Algorithm**

**Input:** 1. PeakFitThreshold - Peak Fitting Threshold [Obtained by Random Sampling]
2. List of Signal profile / Segment

3. **MAXPEAKS** - Upper bound on number of peaks to be fitted for a Segment.

4. **MAXNEDLERTRIALS** – Maximum Nedler Mead trials.

**Output:** Peaks [ Peak Centre, Peak Height, Peak Width, Peak Area ]

---

**Step 1:** Loop through each Segment / Signal Profile.

**Step 2:** Extract X & Y vectors

**Step 3:** for numPeaks = 1 to MAXPEAKS.

**Step 4:** Calculate the initial guess parameters, InitialGuessParams [ Peak Center & Peak Width ]

**Step 5:** Set tuning parameters for Nedler Mead Algorithm.

**Step 6:** for nedlerTrials = 1 to MAXNEDLERTRIALS.

- **Step 6.1:** NedlerTrialParams = NedlerMeadAlgorithm(InitialGuessParams, X, Y, fitgaussian), fitgaussian is the function which the Nedler Mead algorithm optimizes.

- **Step 6.2:** Compute the RMSFitError of the fit with the original data segment.

- **Step 6.3:** If RMSFitError < LowestError, Update LowestError & PeakFitParams.

- **Step 6.4:** Introduce small random variations to InitialGuessParams by Simulated Annealing procedure, to avoid any local minima problem.

**Step 7:** If RMS Error change between \( \text{abs}(\text{LowestError} - \text{LowestErrorFinal}) < \) PeakFitThreshold, Update [ LowestErrorFinal, PeakFitParamsFinal ] and Goto **Step 8** else Goto **Step 3**.
**Step 8:** Output the Peaks [Chromosome, Peak Centre, Peak Height, Peak Width, Peak Area] and Goto **Step 1.**

**Step 9:** End.

The above Pseudocode explains the Gaussian Deconvolution algorithm. It takes as input the peakFitThreshold, MAXPEAKS, MAXNEDLERTRIALS which will be used by the algorithm. **Step 1,** loop through all the Signal Profiles or Segments which contains the following information, Genomic Start Co-ordinate, End Co-ordinate and the BinCount values. **Step 2** extracts the X and Y vectors from Segment. **Step 3** loops through the number of peaks to be fitted starting from 1 to MAXPEAKS. **Step 4** generates initial guesses for all the peaks, mainly Peak Centre and Peak Width and this vector is InitialGuessParams. Peak Width is basically the width of peak at its half height. The Idea is to stitch the peak using its Peak Centre and width. **Step 5** sets some tuning parameters mainly Maximum Iterations and termination tolerance on the function value for the Nedler Mead Algorithm.

**Step 6** loops through the number of Nedler Mead trials. **Step 6.1** starts off the Nedler Mead Algorithm with the input Segment Vectors, InitialGuessParams, function fitgaussian & tuning parameters and gives out the optimum parameters in NedlerTrialParams. **Step 6.2** calculates the Root Mean Square Error of the fit using NedlerTrialParams and the original signal profile say, RMSFitError. **Step 6.3** updates the LowestError[ with RMSFitError ] and PeakFitParams[ with NedlerTrialParams ], if the new RMSFitError is lower than LowestError. **Step 6.4** is the Simulated Annealing procedure responsible for introducing small
random variations to InitialGuessParams which would avoid function to settle at the local minima. Finally, After all the trials, We would have the LowestError and PeakFitParams for a particular number of peaks numPeaks.

Step 7 is responsible to store the final lowest error and the peak fit parameters of all the increasing number of peaks that we are fitting. The step also checks, if this error change within the threshold PeakFitThreshold. If the error change is indeed less than PeakFitThreshold we stop and output the peaks else the algorithm continues with the next numPeaks value. Step 8 stores the peaks into the file with more details about the signal profile.
In Figure 15, the Gaussian Deconvolution is performed on a sample ChIPSeq dataset \([\text{chr1:} 845704-847079]\). It shows 4 plots which represent the different stages of the algorithm on the same Signal profile. The yellow lines represent the Signal profile, the violet Gaussian curves represent the Nedler mead approximation of the individual peaks and the blue curve represents the additive signal resulting from the individual Gaussian peaks. The label below shows the number of peaks fitted and the Root Mean Square Error generated after each fit. The top-left,
top-right, bottom-left and bottom-right plots represent the Gaussian fit of peaks with number 2, 3, 4 & 5 respectively.
Chapter 4: Scalability using MapReduce

This chapter addresses some of the concerns of the scalability which were brought about in the conclusion of the previous section by designing a distributed approach using Google’s MapReduce [7]. Although, much of the Information Retrieval and Data Mining community has embraced MapReduce as a way to solve some of the gripping problems of large scale data processing, Bioinformatics community has been hesitant in this regard.

MapReduce, which is based on a functional programming paradigm, can be applied only to a certain restricted class of problems. There are large class of problems in bioinformatics which involve huge amounts of data and repetitive application of a certain algorithm on them. MapReduce is just the right tool to solve these problems with efficient time.

4.1. Introduction

The sequential processing of the Data using an Algorithm has its limitations. Algorithms can get smarter but only up to a point; the speed with which we can process the data is bound by IO. While processing the data as large as the size of Gigabytes, Terabytes and Petabytes, we need a cluster of computers connected together via high speed network working in tandem, splitting the work and sharing the results between them seamlessly.
In most of the present day large scale computing, the amount of data that can be stored is significantly larger than the amount of data that the machine can work with at an instant of time. Google, Facebook, Ebay and other corporations processes several hundreds of petabytes of data every month, in chunks of data 16GB to 32GB. The average individual Google job size is around 200GB which is a small fraction of what can be stored on a hard disk drive (HDD). Sequential reading of this data would take around 45-50 minutes. This means, we can analyze the data faster than we can read it. Individual high speed SATA drives can only read at 75MB/sec but if we parallelize the reading and have 1000 HDDs working together then we can have a speed of 75GB/sec and thereby catch up with the speeds of CPU’s and RAM. So, Distributed Parallel Computing of the data is inevitable and seems to be the only choice.

4.2. Drawbacks of traditional Distributed Parallel Computing

In traditional distributed systems, the focus was on to get all the systems computing at the same time, but it was not taken in to account where the data was being stored. A lot of libraries were developed such as MPI( Message Passing Interface ), PVM( Parallel Virtual Machine ) etc. In these architecture the data was moved from node to node where computation might occur for which large pipe lines were built between computing stages.

There were some major inherent drawbacks of these approaches which we shall discuss here in general and they are:
Firstly, distributed systems architecture involves splitting the large input data into many small chunks and doing it efficiently is a hard problem. Secondly, When data is shared, lot of inter process communication needs to be managed, which is challenging as it leads to synchronization issues such as deadlocks. The availability of finite amount of bandwidth is also a serious bottleneck; bandwidth is one of the scarce commodities and should be used sparingly. Failures of nodes are common but, for a long process it is desirable to store partial progress and restart in case of a failure. If a node goes down and comes back then there is a lot of state information that needs to be reinstated. It is tough to determine what invariance exists about that state.

The new approach to large scale computing is to move the computing stages to where the data is located rather than moving the data. For computation involving data in the range of hundreds of Terabytes, then architectures like “shared nothing” need to be used where hundreds and thousands of CPUs contribute to individual jobs. In the past the distributed programmers needed to have a high-level algorithm to solve a problem as well as low level knowledge of socket programming. It was desirable to have an architecture where they can
only worry about the problem and leave the low level details to the distributed computing library. These were the motivation behind the development of MapReduce.

4.3. Functional Programming Concepts

Wikipedia describes functional programming as “a programming paradigm that treats computation as the evaluation of mathematical functions and avoids state and mutable data”.

Functional programming is quite different from the modern programming paradigms, procedural and Object oriented. Although, the fundamental ideas have some overlap, the way they are presented, their usage and interpretation are a little different from the rest [11].

4.3.1. Properties of Functional Programming

The main properties of Functional Programming are[12]:

- Functional operations do not modify data structures: They always create new ones.
- Original data still exists in immutable form.
- Sequence of operations does not matter

To understand these properties clearly, we have a few examples below.

**Example 1:** \( \text{fun func1( l: int list )} = \alpha(l) + \beta(l) + \Omega(l) \)
In Example 1, func1 is a function in a functional programming setup. Some of the observations that can be made here are.

- Order in which $\alpha()$ and $\beta()$, etc. are evaluated, does not matter
- They do not modify $l$. New copies of $l$ are created.

**Example 2:** 

```
fun sortAndAddElement( x, list ) =
  Let list' = Sort( list )
  Add ( list' :: x )
```

In Example 2, `sortAndAddElement( )` function sorts a list, adds a new element to the end, and returns that new list. However, In this whole process, it never modifies $list$.

### 4.3.2. Map and Fold Functions

The discussion of Functional Programming is incomplete without the discussion of two of the most important functions $Map$ and $Fold$ and they are explained briefly below.

#### 4.3.2.1. Map Function

$Map$ function is defined as, 

```
map f list: ( a => b ) => ( a list ) => ( b list )
```
In *Figure 16*, *map* function creates a new list by applying $f$ to each element of the input list; and makes sure the output order is maintained[12].

### 4.3.2.2. Fold Function

*Fold* function is defined as,  
\[
\text{fold } f \; \text{list}: (a^* b \rightarrow b) \rightarrow b \rightarrow (a \; \text{list}) \rightarrow b
\]
In Figure 17, fold function traverses across the list, executing $f$ on each element plus an accumulator. $f$ returns the next accumulator value, which is combined with the next element of the list[12]. This happens till we reach the end of list.

4.3.3. Functional Programming and inherent Parallelism

Given a pure functional programming environment, the computation on the elements of a list by map and fold cannot see the effects of the computations on other elements. The idea here is, we can move around computation blocks & apply parallelism, If the computation order of $f$ to individual elements in list is commutative. This is the key that MapReduce programming paradigm exploit[12].

4.4. MapReduce Basics & Architecture

Jeffrey Dean & Sanjay Ghemawat first introduced the concept of MapReduce [7] in their paper, which explained in detail several aspects of processing data in large clusters. Google Inc. has been successfully using it for processing Petabytes of world wide web’s data.
A MapReduce computation takes a set of *input* Key Value pairs, and produces a set of *output* Key Value pairs. The user of the MapReduce library expresses the computation as two functions: *Map* and *Reduce*.

### 4.4.1. Map & Reduce Interfaces of MapReduce

*The Map* function, written by the user, takes an input pair and produces a set of *intermediate* key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key $\alpha$ and passes them to the *Reduce* function[11].

$$\text{map} \ (\text{in\_key, in\_value}) \rightarrow (\text{out\_key, intermediate\_value}) \ list$$

The *Reduce* function, also written by the user, accepts an intermediate key $\alpha$ and a set of values for that key. It merges together these values to form a possibly smaller set of values. Typically just zero or one output value is produced per *Reduce* invocation. The intermediate values are supplied to the user's reduce function via an iterator.

$$\text{reduce} \ (\text{out\_key, intermediate\_value \ list}) \rightarrow \text{out\_value \ list}$$

### 4.4.2. Simple MapReduce Example

*Term Frequency* is a problem from the field of Information Retrieval and Data Mining, which takes a set of documents as input and generates as output as $<\text{Word, Number of occurrences in the whole set of documents}>$. 

44
Pseudocode for Term Frequency:

Map( Text InputKey , String InputValue )


for each ( word w, in InputValue )

   EmitIntermediateKeyValuePair( w , 1 );

Map takes as input, Key Value pair < InputKey , InputValue > and tokenizes the InputValue into words and emits out intermediate Key Value pair < Word , 1 >.

Reduce( Text OutputKey , Iterator < IntermediateCount > )


Set frequency = 0;

for each ( value v, in Iterator < IntermediateCount > )

   frequency = frequency + v;

EmitKeyValuePair( OutputKey , new Text(frequency) );

Reduce takes as input, Key Value pair < OutputKey , Iterator: IntermediateCount > and iterators through all the counts and sums it up and outputs for every word Key Value pair < Word , WordFrequency >.

4.4.3. MapReduce Architecture
In this section, we shall discuss the architecture of MapReduce and also discuss the high level of parallelism that can be achieved from this approach.

![MapReduce Life Cycle Diagram](image)

**Figure 18: Life Cycle of MapReduce Workflow [11]**

MapReduce Life Cycle involves 3 major phases and they are *map* phase, *sort & hash* phase, *reduce* phase. We shall now discuss them in detail with the help of **Figure 18**.

- **Step 1:** Map Phase

  *Map* Phase starts by each nodes reading the part of data from Data Sources local to them [distributed file system data blocks] and starts the execution of map() function and generates intermediate key-value pairs. In **Figure 18**, the map phase is represented by rectangular box, which read their input from data stores and generates the output key-value pairs.

- **Step 2:** Sort & Hash Phase:
Once all the maps are over, *Sort* phase starts off and it is represented in *Figure 18* by horizontal bar called Barrier. Each of the nodes which were involved in the map phase sorts their values and hashes the Intermediate key-value pairs based on their keys. Thus, All the key-value pairs with similar keys, hit single machine thus generating iterator of values for a single key.

- **Step 3: Reduce Phase:**

  *Reduce* Phase runs reduce() function on each Intermediate key-value pairs and they happen in separate nodes. The output key-value pairs are persisted back on distributed file system.

  It is easy to conclude that, *map* phase are running in parallel reading their own subset of data generating intermediate key-value pairs. *Reduce* phase are executed in parallel too, processing their own output key and its values. However, One of the major bottlenecks in the whole workflow is, reduce phase can’t start before all the maps have not been successfully executed [11].

4.5. **Distributed File System**

  Distributed File System(DFS) is one of the most important part of the whole MapReduce architecture. The filesystem that manage storage across a network of machines are called *distributed filesystem*. DFS has some unique features which makes it possible to achieve a more reliable and faster way of data access to map and reduce nodes [10].

47
Some of the key features of DFS are:

- It is used to store large files of the magnitude gigabytes and terabytes.
- It follows the pattern “write once, read multiple times” and is usually stored on hundreds and thousands of commodity hardware.
- It ensures a low latency data access.
- The lowest unit of data storage is called block and they are usually very large in size – 64 MB.
- It supports replication of these data blocks which means, If the replication of DFS is set to 2, then every block is stored twice in the cluster. This helps to recover data in cases of node failures.
Figure 19: Structure of Distributed File System [10]

Figure 19 shows the structure of DFS, with replication set to 2. It contains two major components: datanodes, the nodes which stores data in the form of blocks and namenode, is the node which stores the metadata information of the files, guarantees replication of blocks and load balancing of data blocks. During a map reduce job execution, map() function gets executed on blocks and they happen in the machine where the particular block of a file exists, hence achieving data locality. If the node fails, there is always another copy of the same block elsewhere in the cluster where the map() function can be resumed. The logic works the same
way even for reduce() since the intermediate key value pairs are persisted as data blocks on DFS.

4.6. MapReduce Algorithm for ChIPSeq Deconvolution

In the section, we shall formulate the ChIPSeq problem into a functional programming problem and also design an efficient MapReduce Algorithm which will be discussed in detail. We shall also see how we can obtain scalability benchmarks.

4.6.1. Functional Programming Formulation

In the whole of ChIPSeq Analysis Workflow which was discussed in chapter 3, it turns out that most of the time is spent in Deconvolution step. Deconvolution step involved basically to take the Genomic Segment and breaking them into Gaussian Peaks. The size of Genome makes it difficult to use the above steps as it involves deconvolution of more than a million segments. This step can be done in parallel provided we can put some protocols on the data and how it needs to be organized & read.

The Input to the algorithm are X and Y co-ordinates across the Genome or part of the Genome, where X axis is the Genome Co-ordinate [Eg. Chr1:100, this means the 100th base pair location at Chromosome 1], Y axis is the read count at that Genomic location [Eg. 55. This means, the number of reads overlapping at chromosome 1 at 100th base pair is 55]
So, we have a continuous function formed by long running $X$ and $Y$ co-ordinates across the Genome. So, the input data is divided into Segments which are basically regions of enrichment.

The Idea is to load the input data into DFS as blocks and process these blocks in parallel, in a MapReduce cluster. When the data gets loaded as blocks [ say of size 64 MB ], each block might contain few hundreds of Segments in it. Consider a Segment in block 1 of Node 1 and any other Segment in block 2 of Node 2, It doesn’t really matter in which node we process the Segments. So, Map can indeed be done in parallel and produces peaks out of Segments. The only problem seems to be when a Segment is shared between 2 blocks as they are at the block boundaries. This can be easily solved by Segmentation which will be discussed later in the chapter.

Since, the blocks are split across the cluster, which means the Segments of a particular chromosome are distributed and there must be a way to collate all of them together and run some chromosome wide statistics, which can be easily done in reduce phase. In this phase, all the peaks which belong to a particular chromosome get hashed to a particular node. Consider any 2 chromosome, when peaks of one particular chromosome are processed in Node 1 and the other chromosome in Node 2. This has become inherently parallel as we are interested in chromosomal wide statistics.
Since, We are designing a MapReduce solution, it demands data to be treated as Key Value pairs. We shall define the Key Value pairs at every level and discuss the steps of Algorithm which would solve the problem.

**Figure 20: ChIPSeq MapReduce Workflow**

*Figure 20* displays the complete end to end flow of the ChIP MapReduce workflow. The big rectangle with label as “Source” is the distributed file system where the ChIPSeq Segment data is stored in the form of blocks. The small rectangles with label “M” are Mappers. We have already organized the data as one Segment per line. Mapper reads one line at a time and the Mapper Input Key Value pair is \(<\text{LineNo}, \text{Segment}>\) and it runs the deconvolution algorithm as produces Mapper Output Key Value pair is \(<\text{Chromosome}, \text{Peak}>\). The other small
rectangles with label “R” are Reducers. Since, several mappers might have been acting on a particular chromosomes and producing peaks, Reducer Input Key Value pair is \(<\text{Chromosome}, <\text{Peaks}>\) \(<\text{Peaks}>\) represents iterator of peaks] and runs chromosome wide statistics and emits Reducer Output Key Value pair as \(<\text{Chromosome}, \text{Peak}>\).

### 4.6.2. Main Algorithm of ChIPSeq MapReduce

In this section we shall discuss in detail the algorithm steps involved in the different phases, mainly Map and Reduce.

The Gaussian Deconvolution algorithm discussed in Chapter 3 in detail was converted into a Java API, which mainly has Gaussian Deconvolution Object which implements the Nedler Mead Algorithm for Curve fitting. It has 3 main important functions which the mapper uses.

- \(\text{Init}(x, y)\) – Initializes the Gaussian Deconvolution Object with the Segment.
- \(\text{GaussDeconvoFitting}()\) – Starts the Nedler Mead algorithm and does the Curve Fitting.
- \(\text{getResults}()\) – Gets the results from the Object\([\text{i.e. Peaks}\) that are fitted for Segment.

**Mapper Object**

**Input Key-Value pair:** \([\text{LineNo, Segment}]\)

**Output Key Value pair:** \([\text{Chromosome, Peak}]\)
Step 1: Create a Gaussian Deconvolution Object (say GD) is an engine which drives the fitting Process.

Step 2: Map function gets the Key Value pair as [LineNo, Segment]. Parse the Segment and get the Chromosome and also prepare the X & Y vectors.

Step 3: Initialize GD.Init(X, Y) - This initializes the Gaussian Deconvolution Object and is ready to use for the given Segment.

Step 4: GD.GaussDeconvolutoFitting() - Starts the fitting process and does iterative regression.

Step 5: GD.getResults() - Gets Results[One or more peaks] after the fitting into result matrix.

Step 6: Loop through each row of the matrix and emit key-value pair, <Chromosome, Peak>.

Step 7: Nuke the GD object for reuse and get the next Key Value pair, go to Step 2.

Step 8: End when all the Key Value pairs are processed.

Above Pseudocode is for the Mapper function, All the cluster nodes read their respective data blocks and calls this mapper Object to process the block. One the block is assigned to the node, it is read one line at a time and the Key Value pair [LineNo, Segment] are passed. Step 1 initializes the Gaussian Deconvolution [GD] Object which contains many methods to assist the Deconvolution Process. Step 2, gets the Key Value Pairs and extracts the Chromosome and Segment X and Y vectors. Step 3, Initializes the GD Object with the new segment by passing X and Y. Step 4, starts off the iterative regression by the Nedler Mead minimization algorithm which is inbuilt and generates one or more peaks and puts it in the result matrix. Step 5, fetches the peaks as a matrix where each row represents a peak. Step 6, Loops over the result matrix and emits the output Key Value pair as <Chromosome, Peak>. Step 7, The GD object is
nuked [i.e, not deallocated but prepared for reuse] for the next Key Value pair. Step 8, Program ends when all the Key Value pairs of that block are exhausted.

All the Mappers nodes thus, do processing of their own block completely oblivious of what is happening in the other mapper nodes. After the map phase is over, the shuffle phase starts by sorting of the map outputs and transfers the outputs of map to the reducer nodes as inputs. This ensures that the similar keys are being hashed to a particular node.

Reducer Object

Input Key-Value pair: [Chromosome, <Peak1, Peak2 ... >]

Output Key Value pair: [Chromosome, Peak]

Step 1: Loop through the iterator of peaks and generate chromosome wide statistics.

Step 2: Emit Key Value pair as, [Chromosome, Peak]. Go to Step 1.

Step 3: End when all the Key Value pairs are processed

After the shuffle phase is over, similar keyed map outputs get mapped to a particular reducer node. The Reducer Input Key Value pair is [ Chromosome , <Peak1, Peak2,....> ]. Step 1, loops over the iterator of values and does some chromosome level statistics like average height,
width of peaks which will be later used in the statistics. It also emits the Key Value pair as \([ Chromosome, Peak \)] as output.

### 4.6.3. Conclusion

In case, of a failure or slow execution of any node either mapper or reducer, Job tracker which monitors the whole execution by pinging the nodes and keeping their progress report, kicks off another process in another node which has the same copy of the block exists (for mappers) and there is a new reducer chosen and the key value pairs are channeled to the new node. Thus reliability and fault tolerance is ensured. This is how MapReduce achieves the highest level of parallelism and reliability.
Chapter 5: Results

This chapter discusses the important results at different stages of the ChIPSeq pipeline. The data set chosen was Chromosome 1 and other subsets of Genome from the DoxP53 experiment.

5.1. BinCount Algorithm Results.

BinCount Algorithm is one of the most important step of the whole pipeline which identifies the enriched region and pruning the regions which might not be statistically and biologically significant in our analysis.
Figure 21: BinCount Algorithm Simulation with different sized bins.

Figure 21 is the simulation of the BinCount algorithm with different bin sizes across the genome. The X-axis represents the Numbers of Reads/ Bin and Y-axis represents the Logarithm of Number of Bins. The simulation was done with bin width of 50, 100, 200, 250 with minimum reads required in a bin, set to 5. The most important observation was lower the bin size, more bins were pruned from the analysis because of the hard restriction that we expect at least 5 reads in a particular bin width. The bin width chosen for analysis was 200. Since, It gave a perfect balance between pruning the regions and selecting the enriched regions.

5.2. Intersection of Deconvoluted Peaks with TFBS

After the Gaussian deconvolution of the Segments, the final result are the peaks. As discusses before, These peaks are the direct representation of the TFBS and also to an extend quantify the quality of the TFBS. The peaks are then intersected with the known TFBS to see the quality of the peaks.

The main idea here is to see how many of the known TFBS locations are close to the peak center and within a certain threshold of distance. Ideally, We would want only one TFBS per peak and it has to be as close to the peak center as possible. In the analysis, three TFBS nearest to the peaks are considered and distance metrics are generated and evaluated. The analysis was
done with 200, 100 and 50 base pairs (bp) distance from the peak center which would provide interesting insights into the performance of the deconvolution algorithm.

5.2.1. Results with 400bp window from Peak Center

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3591</td>
<td>388</td>
<td>48</td>
<td>14</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

*Table 1: Results with 400bp window from the Peak Center*

In Table 1, The first row represents the number of TFBS within the 400 bp vicinity (flanking 200 bp on either side of the peak center) and the second row represents the count of peaks with that distance.

The main observations from Table 1 are:

- Average TFBS/Peak: **1.1398**
- Average First Near TFBS Distance: **85.42 [#4031]**
- Average Second Near TFBS Distance: **101.21 [#457]**
- Average Third Near TFBS Distance: **109 [#69]**

In Figure 22, The plot is between TFBS and peaks with TFBS Score in Y-Axis and Peak Height in X-Axis. TFBS Score is a value between 4 and 18 and is a representation of the quality of the TFBS. The main observation in this plot is that, there are many smaller peaks
generated from the Gaussian deconvolution algorithm and which might be of no interest to us. However, the peaks with better height mostly have a high scoring TFBS. However, these results are for TFBS within 400 bp vicinity of the peak center and might result in a lot of false positives. An interesting variation would be to see how these peaks and TFBS vary with the shrinking of the window, which is shown in the next section.

Figure 22: Intersection of TFBS and Peaks with 400bp window

### 5.2.2. Results with 200bp window from Peak Center

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>2219</td>
<td>188</td>
<td>19</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Table 2: Results with 200bp window from the Peak Center

The main observations from Table 2 are:

- Average TFBS/Peak: **1.1104**
- Average First Near TFBS Distance: **44.7186 [ #2420 ]**
- Average Second Near TFBS Distance: **48.5275 [ #218 ]**
- Average Third Near TFBS Distance: **50.0667 [ #30 ]**

Figure 23: Intersection of TFBS and Peaks with 200bp window

In Figure 23, the plot is between Peak Height and TFBS Score. The main observation here is that most of the good peaks with significant height and width have a high scoring TFBS under them and they are within 100 bp vicinity.
5.2.3. Results with 100bp window from Peak Center

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>6</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1318</td>
<td>100</td>
<td>8</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

*Table 3: Results with 100bp window from the Peak Center*

The main observations from *Table 3* are:

- Average TFBS/Peak: **1.1012**
- Average First Near TFBS Distance: **24.28** [ #1416 ]
- Average Second Near TFBS Distance: **26.8609** [ #115 ]
- Average Third Near TFBS Distance: **28.00** [ #15 ]
In Figure 24, the plot shows that peaks with good height and width have their nearest TFBS within the 100 bp window. This is an important result as the confidence in the predicted peaks are very high since we just proved that the known binding sites are in good congruence with the peaks that have been predicted by our algorithm.

5.3. Overall Peak Prediction

In the previous section, the quality of the peaks generated was validated against the known binding sites and the results looked promising. However, that was mainly for the peaks that
intersected with the TFBS. There is another possibility where the peaks are not intersected with the TFBS but have a good height and width and they are also important.

In Figure 25, the histogram plot is performed with Peak Height in X-axis and # of Peaks in Y-Axis of all the peaks. The red line represents the histogram and the black vertical line represents the threshold of the peak height (Say x%) above which the peaks are considered and the rest are pruned from our analysis. The detailed analysis for different window width and varying x threshold is shown below.
In Figure 26, there are 3 major analysis which is for different window width flanking both sides of the peak center. The x% is varied for each window and the different height thresholds are calculated. Column 1 represents the x%, Column 2 represents the corresponding height obtained from the x% of maximum peak height, this helps us to remove the excessive offsets in height. Column 3 represents the number of peaks above that threshold. Column 4 & 5 represents the number of peaks with TFBS and their overall percentage. Column 6 & 7 represents the number of peaks without TFBS and their overall percentage.
The main observation in the above analysis is with the increase in the height threshold, the number of peaks with TFBS in them is increasing drastically. So, the predicted peaks with good height show confidence with the already known binding sites.

5.4. Comparison with MACS

This section compares the results of the Gaussian Deconvolution algorithm with MACS[2].

Figure 27: Peaks from ChIP Gaussian Deconvolution Algorithm
Figure 27 is a plot of the Peak Height in $X$-axis and the Nearest Tfbs Distance on $Y$-Axis. Each green dot represents a peak and its position represents the distance of the nearest TFBS from the center line which is actually the peak center. It can be seen that the best peaks are all clustered near the center line. Ideally, we would want them to be as close as possible to the center line.

![Figure 27: Peaks from MACS Algorithm](image)

Figure 27: Peaks from MACS Algorithm

Figure 28 is a plot of the Peak Height in $X$-axis and the Nearest Tfbs Distance on $Y$-Axis. Each red dot represents a peak and its position represents the distance of the nearest TFBS from the center line which is actually the peak center. The peaks obtained from the MACS
algorithm are scattered from the peak center and they are not as closely spaced as the Gaussian Deconvolution algorithm.

The Figure 29 below shows the juxtaposition of the peaks from both the methods and it can be clearly seen that Gaussian Deconvolution performs much better in predicting the center of the peak than MACS since the peaks are much nearer to the center line which is the peak center. The only drawback of ChipGaussDecon algorithm seems to be the number of peaks predicted with small height.
Figure 29: Juxtaposition of peaks from Gaussian Deconvolution & MACS

Table 4 shows the analysis of the two algorithms Gaussian Deconvolution (Blue) and MACS (Red). The peaks are divided into peak height ranges and statistics are generated which mainly includes # of peaks with TFBS and Average Nearest TFBS Distance.

<table>
<thead>
<tr>
<th>Height of Peaks</th>
<th>No of Peaks with TFBS</th>
<th>Average Nearest TFBS Distance</th>
<th>Height of Peaks</th>
<th>No of Peaks with TFBS</th>
<th>Average Nearest TFBS Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;250</td>
<td>1</td>
<td>10</td>
<td>&gt;250</td>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>200-250</td>
<td>9</td>
<td>10.33</td>
<td>200-250</td>
<td>5</td>
<td>11.8</td>
</tr>
<tr>
<td>150-200</td>
<td>6</td>
<td>17</td>
<td>150-200</td>
<td>5</td>
<td>29.4</td>
</tr>
<tr>
<td>100-150</td>
<td>20</td>
<td>13.9</td>
<td>100-150</td>
<td>13</td>
<td>20.46</td>
</tr>
<tr>
<td>50-100</td>
<td>51</td>
<td>20.47</td>
<td>50-100</td>
<td>51</td>
<td>31.5</td>
</tr>
<tr>
<td>30-50</td>
<td>62</td>
<td>32.61</td>
<td>30-50</td>
<td>70</td>
<td>34.15</td>
</tr>
<tr>
<td>10–30</td>
<td>339</td>
<td>54.71</td>
<td>10–30</td>
<td>235</td>
<td>53.49</td>
</tr>
<tr>
<td>5–10</td>
<td>914</td>
<td>81</td>
<td>5–10</td>
<td>110</td>
<td>64.7</td>
</tr>
</tbody>
</table>

Table 4: Analysis between Gaussian Deconvolution & MACS
In Figure 30, the plots show some detailed analysis of the 2 algorithms with peak ranges. The left figure is a plot of Peak Height Ranges on X-axis with Number of Peaks on Y-axis. Both the methods are well in sync with the peaks with good heights whereas ChipGaussDecon suffers when the peak height decreases and the main reason for this is during the deconvolution step, the algorithm produces a lot of small peaks with small height especially when it cuts the segment shoulders. However, these can be strategically pruned and this will be explained at a later stage.
The right figure is a plot of Peak Height Ranges on X-axis with Average Near TFBS distance of Peaks on Y-axis. ChipGaussDecon algorithm outperforms the MACS for the peaks with good height when the average Near TFBS distance of the peaks is measured for different Peak Height ranges. However, It deteriorates when it reaches the peaks with smaller heights. But, This problem can be avoided by strategically pruning the peaks with smaller height. One of the main techniques is to remove the smaller peaks in a segment or enriched region if there is a peak with greater height already in the vicinity of the TFBS. However, It can be part of the future work.

5.5. MapReduce Results

This section discusses the results obtained after addressing the scalability concerns of the Gaussian Deconvolution algorithm. The whole genome data of the ChIPSeq experiment was taken, was preprocessed and pushed into the Hadoop File System (HDFS). Amazon EC2 and Elastic MapReduce were used to bring the nodes on cloud on demand and to run the Hadoop jobs.
In Figure 31, The plot is between Number of Nodes is Hadoop Cluster in X-axis and Time (in Minutes) in Y-axis. The observe the performance in terms of time, the size of the cluster was varied gradually and it was observed that there was a liner decline in the time as the number of nodes was increased.

A more detailed analysis is shown below:

- 20 Nodes [1 Master 19 Slaves] : 1 Hour 24 Mins
- 18 Nodes [1 Master 17 Slaves] : 1 Hour 35 Mins
- 17 Nodes [1 Master 16 Slaves] : 1 Hour 42 Mins
- 15 Nodes [1 master 14 Slaves] : 1 Hour 54 Mins
- 13 Nodes [1 Master 12 Slaves] : 2 Hours 8 Mins
**Baseline:** 27.97 Hours, When run sequentially.

For each of the simulations, a minimum of 3 runs were made and the average time was taken into consideration. A typical MapReduce process has 3 main steps: Map phase, Shuffle phase and the Reduce phase. In all the simulations, the total time distribution was recorded and it was observed that the Map phase takes 95%, Shuffle phase – 5% followed by Reduce phase – 5% of total time. So, It can derived that, with further increase in number of nodes in cluster, the timing can be bettered.

The most important advantage of MapReduce is that the algorithm needs no change when we are increasing or decreasing the number of nodes. The algorithm itself is oblivious of the changes in the cluster node structure & size.
Chapter 6: Conclusions & Future Work

6.1. Introduction

In this chapter, we will conclude the thesis and give research directions for future work.

6.2. Conclusion

In this thesis, we have provided a comprehensive pipeline in analyzing the ChIPSeq dataset. It has many modules in a sequence and the most important modules being the BinCount Algorithm and the Gaussian Deconvolution algorithm. BinCount Algorithm reduces the search space for TFBS by intelligently pruning the genomic regions and by providing the enriched regions. One of the major contributions to the thesis is the Gaussian Deconvolution algorithm which addresses the problem of closely spaced binding sites by splitting the segments into Gaussian peaks. The comparison of the results with the best ChIPSeq software currently, MACS shows that our algorithm performs way better in many aspects.

One of the main concerns of our algorithm was the scalability and this arises from the iterative curve fitting procedure which the Gaussian Deconvolution algorithm uses at its core. We addressed this issue by mapping the problem to a completely new domain of functional programming. Once the problem was successfully mapped to the functional programming domain, we used the distributed data processing method MapReduce to process the data in...
parallel. The data were processed on Amazon Cloud and we achieved good benchmarking in terms of timing. We were also able to show, how the algorithm scaled linearly with the increase in the number of nodes in the cluster.

6.3. Future Work

This thesis was one of the first step towards the suggestion of Peak Deconvolution which Pepke et al. suggested in their survey paper of different implementations to analyze ChIPSeq[5]. A more sophisticated statistical modeling can be developed on which the Gaussian deconvolution can be applied. We also feel, the analysis of the peaks after their generation can be much more sophisticated than the current existing analysis which still gives us decent results. Currently, Only the part of the algorithm, the deconvolution is done in the cloud. But, the complete automation of the algorithm in MapReduce can be of great help to the research community. Finally, Bio Informatics domain should increasingly consider cloud based solutions for some of the problems which can be mapped to functional programming domain. Since, this would allow them to process more data than before, use computationally expensive but precise algorithms & perform rigorous analysis on data within a short period of time.
References


[9] **An Introduction to Signal Processing in Chemical Analysis.**
   *Link:* [http://terpconnect.umd.edu/~toh/spectrum/InteractivePeakFitter.htm](http://terpconnect.umd.edu/~toh/spectrum/InteractivePeakFitter.htm)
    Link: http://developer.yahoo.com/hadoop/


    Link: www.code.google.com/edu/submissions/mapreduce