HiDEW: High-speed Determination of Epigenetic code rendered through a Web application

by

Ankit Arvind Arte

(Under the direction of Jonathan Arnold)

Abstract

Histone modifications, especially methylations, play a vital role in transcriptional activation of genes within a human body. Hence, we created a web service to help users to study the local spatial structure of these modifications along a chromosome along with their patterns. We created four queries for this purpose. A memory mapping technique along with binary search was used to help obtain quick output results of these queries. We also created a tree representation to view the homogenous regions of these modifications as we walk through the chromosome. We not only show there is a large non-uniformity of the spatial distribution of epigenetic marks in these regions, but also that there are a large number of regions which are locally multinomial.

Index words: High-Speed queries, Epigenetic code, Human genome, Histone modifications, Homogenous regions within a chromosome

HiDEW: **H**igh-speed **D**etermination of **E**pigenetic code rendered through a **W**eb application

by

Ankit Arvind Arte

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

masters of science

Athens, Georgia

2011

HiDEW: High-speed Determination of Epigenetic code rendered through a Web application

by

Ankit Arvind Arte

Approved:

Major Professor: Jonathan Arnold

Committee: Krzysztof Kochut
Rodney Canfield

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
May 2011

HiDEW: **H**igh-speed **D**etermination of **E**pigenetic code rendered through a **W**eb application

Ankit Arvind Arte

May 2011

Acknowledgments

I gratefully acknowledge my advisor, Dr. Jonathan Arnold, whose expertise in this field helped me improve my knowledge and writing abilities. This thesis would not have been possible without him.

I extend a special thanks to my committee members Dr. Krzysztof Kochut and Dr. Rodney Canfield who helped me work through this thesis.

This work was supported by NSF grants DBI-0646315 and BES-0425762.

I would also like to thank my friends for the encouragement they have given me throughout this thesis.

Finally, I would like to thank my parents, who always have been a source of inspiration and support. I can never thank them enough for their love and sacrifice.

Contents

List of Figures

Chapter 1

Preface for the Readers

1.1 Human Genome

Every living organism, including human beings, has a genome that comprises of all the biological information needed for the development and functioning of that organism. A human genome is subdivided into chromosomes which are present in the nucleus of every cell. In humans, there are 22 pair of chromosomes plus a pair of sex chromosomes (XX in females and XY in males). Each chromosome is made up of Deoxyribonucleic acid, DNA, which contains the biological information. This DNA molecule encodes genes, which consists of proteins that direct how a human body is built and maintained. A human genome consists of about 35,000 genes.

1.2 DNA

DNA is a double-stranded chemical structure, also known as a double-helix. The two strands in the double-helix are made up of an alternating sugar and phosphate backbone running parallel to each other, but in opposite directions. DNA is also made up of four chemical bases:

adenine (A), guanine (G), cytosine (C), and thymine (T). It is the sequence of these bases, that determine the information available for the development and maintaining a human body. These bases pair up with each other, A with T and C with G, via hydrogen bonding, forming a base pair and are the steps o the helical staircase. On an average, a chromosome is 140 million base pairs long.

The DNA is wrapped around proteins known as histones. There are four primary types of histones: H2A, H2B, H3 and H4, as well as other variants like H2AZ. There are two copies of each of these histones, hence forming an octamer.

An overview of sections 1.1 and 1.2 is shown in figure 1.1

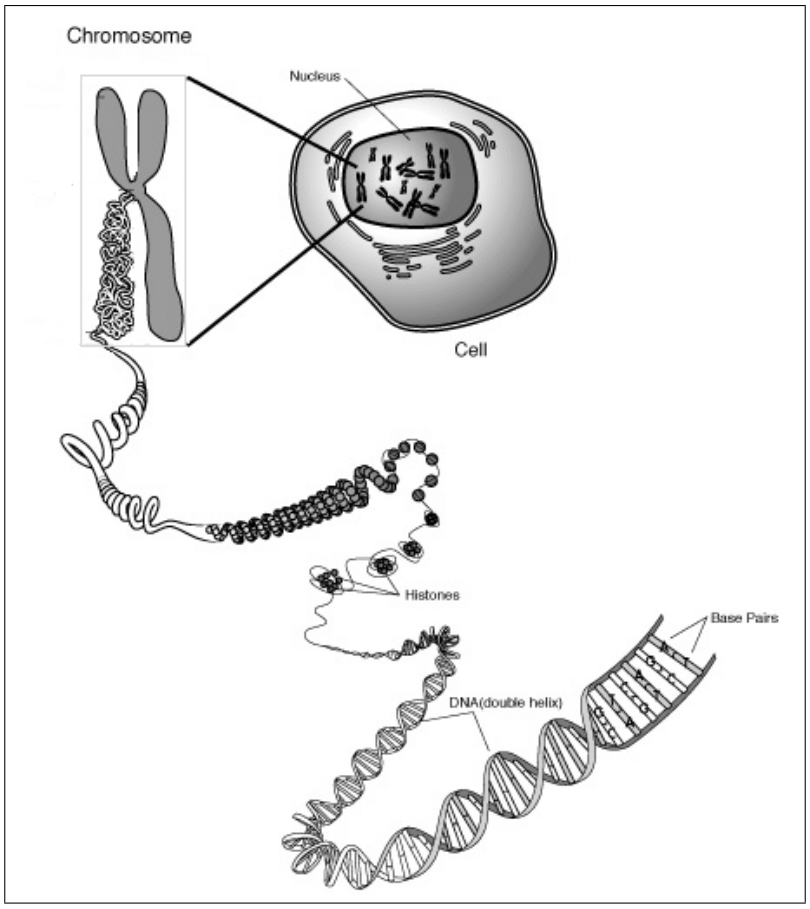


Figure 1.1: Overview of Chromosome and DNA in human body
Source: http://creationwiki.org/File:Chromosome.png

[h]

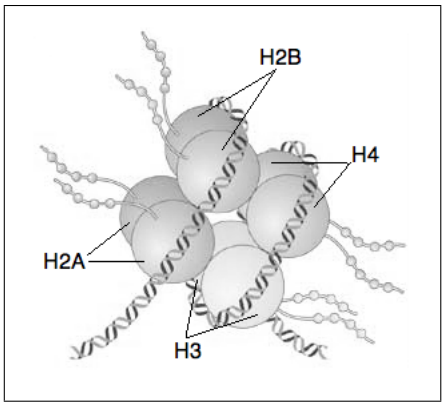


Figure 1.2: Four Core Histones
Source: http://www.nature.com/nrn/journal/v6/n2/fig_tab/nrn1604_F2.html

1.3 Histone Modification

Each of the four core histones has a tail which protrudes out of a nucleosome as shown in the figure 1.2. This tail is subject to a variety of chemical modifications because of the amino acid that is present in a modified histone protein. These modifications include acetylation, methylation, phosphorylation, ubiquitylation, and many more. The kind of modification and its location in a DNA sequence is called the epigenetic code. This epigenetic code in part decides how we look and who we are.

Among these methylations are a kind of modification which adds one or more methyl groups to an amino acid. Methylations are important because adding of a methyl group results in the "start" of a reaction which can eventually lead to the formation of a protein. Here, in this thesis, we study methylation modification.

1.4 DNA Sequencing and Pattern Search

The term DNA Sequencing is defined as the precise determination of the order of the bases - adenine, guanine, cytosine and thymine in a sample of DNA.

For example:

CATAAACCCTGGCGCGCTCGCGGGCCGGCACTCTTCTGGTCCATGG is a part of a DNA sequence.

The epigenetic code is determined by a new high-throughput method of genome sequencing.[4]

A pattern is defined as a cluster where nucleotides appear on the DNA. Hence a DNA Pattern Search accepts a search pattern and returns the number and positions of all the sites that match the pattern.

For example:

A pattern search for pattern 'CTACGATC' might return the following:

Match#1 from position 170 to 178

Match#2 from position 8000 to 8008

Match#3 from position 123861 to 123869

Match#4 from position 35638234 to 35638242 and so on

A review of how to do this can be found in[8]. This closely is related to the problem of finding epigenetic motifs. An epigenetic motif can be defined as a part of a DNA sequence which is recurring at various positions over the chromosome.

1.5 Objectives of this thesis

Barski et. al [1] studied twenty three histone modifications using a new Solexa 1G sequencing technology, and their data can be downloaded from [5]. We aim at determining a local spatial

structure in the occurrences of these modifications as we walk along a chromosome. To ascertain this, it is necessary to create a data structure that allows queries about the spatial locations of histone modifications in a DNA sequence quickly. Once this data structure is created, we can ascertain the spatial structure of the epigenetic code along a chromosome.

Chapter 2

Introduction

A human genome consists of 23 chromosomes, about 140 million base pairs long, on an average, and made up of DNA. DNA is wrapped around nucleosomes which are comprised of 4 core histones(H2A, H2B, H3 and H4). Histones are essential protein components of a chromatin and play a vital role in gene regulation. The amino-terminus of the core histones protrude out of the nucleosome. This is called a 'histone tail', and is subject to various post-translational modifications such as acetylation, methylation, phosphorylation, ubiquitylation, etc. The kind of modification and its location in a DNA sequence is called the epigenetic code. This epigenetic code determines in part how we look and who we are.

Among these modifications, methylations at lysine and arginine residues are very stable and are important for transcriptional activation of genes. Histone methylation is the modification of certain amino acids in a histone protein by the addition of one, two, or three methyl groups. There are many kinds of modifications that can be assayed in a genome.

Twenty three such histone modifications were studied closely by Barski et al.[1] using Solexa 1G sequencing technology.These include H2BK5me1, H3K4me1, H3K4me2, H3K4me3, H3K9me1, H3K9me2, H3K9me3, H3K27me1, H3K27me2, H3K27me3, H3K36me1, H3K36me3, H3K79me1, H3K79me2, H3K79me3, H3R2me1, H3R2me2, H4K20me1, H4K20me3, H4R3me2.

Along with these, histone variant H2A.Z, RNA polymerase 2 and insulator binding protein CTCF were also examined.

The nomenclature for these modifications is as follows:

1. The name of the histone (e.g. H3)
2. The single letter amino acid abbreviation (e.g. K for Lysine, R for arginine) and the amino acid position in the protein, thereby specifying where the modification occurs on the histone.
3. The type of histone methylation (me1: mono-methylation, me2: di-methylation, me3: tri methylation).

So H3K4Me1 denotes the mono-methylation of H3 on the 4th lysine from the start (N-terminal) of the histone protein.

Our task is to determine whether or not there is local spatial structure along a chromosome in the occurrence of these modifications. This work is motivated by [6]. To ascertain this, it is necessary to create a data structure that allows queries about the spatial location in a DNA sequence of histone modifications quickly.

Hence we created a data structure that consists of 3 parts - position, mark and count.

position is the co-ordinate along the chromosome.

mark is a unique one letter code for the modification.

count is the number of modifications at a particular position. In other words, the count is the number of DNA sequence reads that cover a particular position on the human genome. To save space, we used a unique one letter code, a single alphabet, for each of the 23 modifications.

The chromosome data are stored in these data structures and written into files in a binary format, which are then stored on a Sun workstation as .bin files. As this project studies the pattern of modifications along each chromosome, the data from the original data files were

broken down chromosome-wise which resulted in the creation of 24 binary files. (one for each chromosome, including chromosomes X and Y).

As these datasets are very large, there is a need to find an efficient way of accessing and retrieving the data. The binary files were memory mapped for this purpose. This resulted in quick recovery of the required data, as the memory map helps in array-like access to the data, which means that any row can be accessed directly by specifying its position on the array. This increased the speed tremendously which culminated in very quick queries.

Four queries were developed to help the users to study the spatial placement of these modifications on a chromosome. These queries then enabled the construction of graphical tree representation of spatial distribution of modifications along each chromosome, where the terminal nodes denote homogeneous subintervals on the chromosome. Within these subintervals, the distribution of 'modifications' appears locally multinomial.

Chapter 3

Methods and Materials

3.1 Data Source

The data used here can be downloaded from [5]. The data contains the 23 modifications, each having its corresponding reads over a DNA sequence of the human genome, randomly ordered, for all the chromosomes over the human genome.

3.2 Hardware

The web application which is developed for users to run the queries and view the distribution of modifications graphically is deployed on a Sun Solaris OS version 5.10 machine with an intel i386 processor.

MySQL and JDBC were used on a localhost having Mac OS X version 10.6.4 with 2.26 GHz Intel Core 2 Duo processor and 2 GB RAM.

3.3 Software

The web application has been hosted on an Apache2 Web sever. The front-end was developed in HTML and JavaScript. ClearSilver version 0.10.5 templating system was used for creating HTML templates to display the results to the users as per their input queries. Using these templates simplified the process of making any future changes to the HTML front-end, without affecting the rest of the application.

The 4 queries were coded in C++ version 3.4.3. These programs memory map the binary files, stored on the server. CgiCC version 3.1.5 has been used as an interface between the queries and the front-end HTML.

Java version 1.6.0_20 and MySQL version 5.5.3 were used heavily in the initial stages of this project. As the original files are very large, there was a necessity for an efficient way of storing these files. MySQL database was used initially for this purpose, but then was abandoned in favor of the binary files with data structure described in section 2.5. JDBC (Java Database Connectivity) API was used for finding coverages, run lengths, spacing and correlation coefficients.

3.4 MySQL Database

Early computation led to the creation of 24 MySQL tables, one for each chromosome.Each table consisted of all the modifications on that particular chromosome. Mainly, each table had 3 columns - position, mark and count. As these tables were large, above 100 million rows, high latency was observed when running queries on them. For this reason, the idea of using a MySQL database was discarded, and the binary files-memory mapping approach was adopted for efficiency and speed.

3.5 Data Structure

```
typedef struct {
    unsigned int pos;
    char mark;
    short int count;
} Marker;

Marker *rows;
```

Figure 3.1: Structure used for storing the data

Figure 3.1 shows the data structure "Marker" that has been used to store the data of a chromosome.

The data structure is of type 'struct' and consists of 3 datatypes to store the chromosome data - pos, mark and count. "pos" is of type 'unsigned int' since we know that positions don't have negative values. "mark" is of type 'char' since we have associated every modification with a single letter. "count" is the number of modifications of a particular type on that position of the chromosome and is of type 'short int'.

Every line of data in the file consists of the above mentioned three values and hence can be directly inserted into the structure.

An array of structs was created to store the necessary data, which was then written into a binary file. This process was carried out for each of the 24 chromosomes, thus creating 24 binary files with rows of data each containing the 3 data types. The binary files are then accessed and read in the query programs to generate a memory map for quick access of this data. "rows" is the pointer to an array of 'Marker' structs, as shown in fig 3.1. It points to the memory mapped contents from the binary file.

3.6 Queries

Four queries were created to help the user to observe the spatial pattern of modifications. Since the application created is a web-based application, the emphasis of this project was as much on speed as it was on the correctness of the results. Since, there was a huge amount of data to be queried, running large queries on this data was a slow process. This led to the creation of binary files with memory mapping for easy and quick access to the data.

To implement these queries, it was necessary to use a search algorithm which can quickly search for a particular position in a chromosome. Since the data is an ordered list of positions, binary search was opted for this process. Generally binary search returns the position of the key value being searched if it is present and a NULL value, if the key is not present. In this project however, there was a need to get a range of data between the start and end positions entered by the user. Hence, the nearest position to the key needs to be returned, if the key is not present in the data. The traditional algorithm of binary search was thus tweaked, so that it returned the nearest position, greater than the start position requested by the user.

The following is the pseudo code for the binary search used here.

begin

```
left = 0
right = number of lines in the file
key = value to be searched
while left ≤ right do
   mid = (left + right)/2
   if key == (rows[mid]→position)  then
      return mid // found the value
   else if (key > (rows[mid]→position) then
      left = mid +1
   else
      right = mid +1
   end if
end while
// if not found
```

```
  // move pointer to the nearest value greater than key
  if (rows[mid]→position) ≥ key then
    return mid;
  else
    return (mid+1)
  end if

end
```

3.6.1 Query1: Find all (x,y)

The first query is implemented to allow users to be able to search between any two positions(x and y) in a DNA sequence for histone modifications. This is a basic query which will return the positions, modifications and their counts.

The pseudo code of this query is as follows

```
  begin
  //take user input values for the interval he wants to search
  start = start position of the interval
  end = end position of the interval
  results = vector for holding the output
  rows = memory_map(filehandle)
  // call the binary_search function
  num_lines = number of lines in the file
  bin_pos = <Binary_Search>(0, num_lines-1, start)
  // due to repeated positions in the binary file,
  // binary search returns the position of the first match
  // move back until you encounter the first occurrence of the position being
  //searched
  while bin_pos ≥ 0 do
    if rows[bin_pos]→position ≥ start then
      decrement bin_pos
    else
      break
    end if
  end while
  found_count = 0; // counter for number of positions found in the range
  while (rows[bin_pos]→position) ≤ end && bin_pos ≤ (num_lines-1) do
    increment found_count
    insert rows[bin_pos]→position into results
```

```
    increment bin_pos
  end while
  if found_count == 0 then
    print "No modifications found"
  else
    display results //display positions,marks, and counts
  end if
end
```

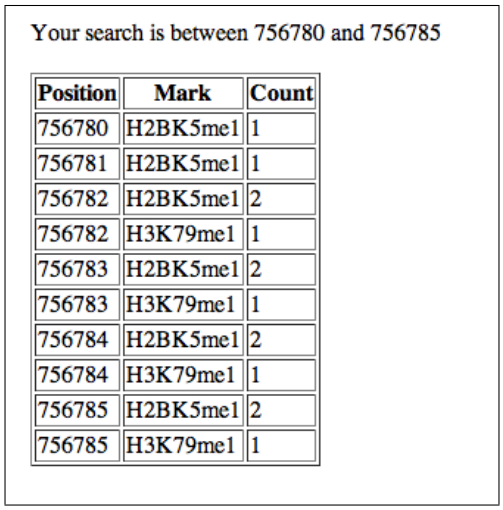


Figure 3.2: Screenshot of first query

Figure 3.2 shows a screenshot of the first query when searched between DNA positions 756780 and 756785.

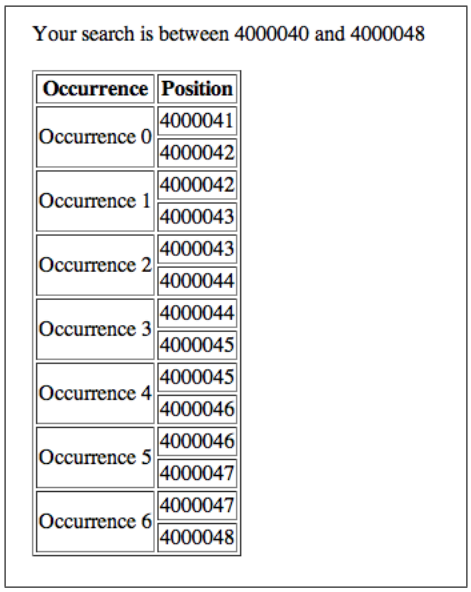
3.6.2 Query 2: Find p-mer in (x,y)

The second query enables users to a find a sequence of modifications over a particular interval. We called this a 'p-mer'. A 'p-mer' is defined as a *particular sequence of modifications over a chromosome in a consecutive order*. For example a p-mer such as *'cst'*, in its one letter code, is a tri-mer since it consists of 3 histone modifications, c, s and t respectively for H2BK5me1, H3R2me2, and H4K20me1. All the modifications along with their one letter code can be viewed on the website. The following is the pseudo code for this query

```
 begin
//take user input values for the sequence he wants to search
start = start position of the search interval
end = end position of the search interval
mods = sequence of modifications
inputs = vector for holding the modifications entered by user
pos = vector for storing single matched occurrence
inputs.push_back(mods)
filehandle = open(binary file)
rows = memory_map(filehandle)
// call the binary_search function
num_lines = number of lines in the file
bin_pos = <Binary_Search>(0, num_lines-1, start)
// due to repeated positions in the binary file,
// binary search returns the position of the first match
// move back until you encounter the first occurrence of the position being
//searched
while bin_pos ≥ 0 do
  if rows[bin_pos]→position ≥ start then
    decrement bin_pos
  else
    break
  end if
end while

// counter for number of occurrences of the particular input pattern
occ_count = 0
// counter for number of positions found in the range
found_count = 0
num_inputs = size of inputs
while (rows[bin_pos]→position) ≤ end && bin_pos ≤ (num_lines-num_inputs) do
  // store occurrence position since we need to list multiple occurrences
  prev_occ = rows[bin_pos]→position
  for j = 0 to num_inputs do
    mark_string = collect all marks from position //same as rows[bin_pos]→position
    non_inputs = characters other than input marks
    if (mark_string contains inputs[j]) && (mark_string does not contains non_inputs)
    then
      increment found_count
      insert rows[bin_pos]→position into pos
    end if
  end for
```

```
    if found_count = num_inputs then
      // all input marks matched in the occurrence
      insert all values from pos into occurrence[occ_count]
      increment occ_count
    end if
    increment bin_pos // until next position value is encountered
  end while
  if occ_count == 0 then
    print "No modifications found"
  else
    display occurrences //display only positions
  end if
end
```



Your search is between 4000040 and 4000048

Occurrence	Position
Occurrence 0	4000041
	4000042
Occurrence 1	4000042
	4000043
Occurrence 2	4000043
	4000044
Occurrence 3	4000044
	4000045
Occurrence 4	4000045
	4000046
Occurrence 5	4000046
	4000047
Occurrence 6	4000047
	4000048

Figure 3.3: Screenshot of second query

Figure 3.3 shows the result of the second query when searched for p-mer 'fk' between the interval 4000040 and 4000048. It shows that there are 7 occurrences of this p-mer in the specified interval.

3.6.3 Query 3: Find p-mer $\geq$ Z in (x,y)

This query will allow users to search for a p-mer which have counts above the specified threshold value,Z. In other words, Z is the number of reads associated with a modification, at a defined position in a DNA sequence.

The following is the pseudo code for this query

begin
//take user input values for the sequence he wants to search
start = start position of the search interval
end = end position of the search interval
z=depth of the stack
mods = sequence of modifications
inputs = vector for holding the modifications entered by user
pos = vector for storing single matched occurrence
inputs.push_back(mods)
filehandle = open(binary file)
rows = memory_map(filehandle)
// call the binary_search function
num_lines = number of lines in the file
bin_pos = <Binary_Search>(0, num_lines-1, start)
// due to repeated positions in the binary file,
// binary search returns the position of the first match
// move back until you encounter the first occurrence of the position being
//searched
while bin_pos $\geq$ 0 **do**
  if rows[bin_pos]→position $\geq$ start **then**
    decrement bin_pos
  else
    break
  end if
end while

// counter for number of occurrences of the particular input pattern
occ_count = 0
// counter for number of positions found in the range
found_count = 0
num_inputs = size of inputs
while (rows[bin_pos]→position) $\leq$ end && bin_pos $\leq$ (num_lines-num_inputs) **do**
  // store occurrence position since we need to list multiple occurrences
  prev_occ = rows[bin_pos]→position

```
    for j = 0 to num_inputs do
      mark_string = collect all marks from position //same as rows[bin_pos]→position
      non_inputs = characters other than input marks
      if (mark_string contains inputs[j]) && (mark_string does not contains non_inputs)
      && (rows[bin_pos]→count) ≥ Z) then
        increment found_count
        insert rows[bin_pos]→position into pos
      end if
    end for
    if found_count = num_inputs then
      // all input marks matched in the occurrence
      insert all values from pos into occurrence[occ_count]
      increment occ_count
    end if
    increment bin_pos // until next position value is encountered
  end while
  if occ_count == 0 then
    print "No modifications found"
  else
    display occurrences //display only positions
  end if
end
```

Figure 3.4 shows the result of the third query when searched for p-mer 'cst' with Z $\geq$ 2 between the interval 2930 and 2940. It shows that there are 6 occurrences of this p-mer in the specified interval.

3.6.4 Query 4: Find p-mer in (x,y) and not in (r,s)

This query allows users to find a p-mer in a interval(x,y) but not in another interval(r,s).

The following is the pseudo code for this query

```
  begin
//take user input values for the sequence he wants to search
start = start position of the search interval
end = end position of the search interval
r = start position of the second search interval
s = end position of the second search interval
mods = sequence of modifications
inputs = vector for holding the modifications entered by user
```

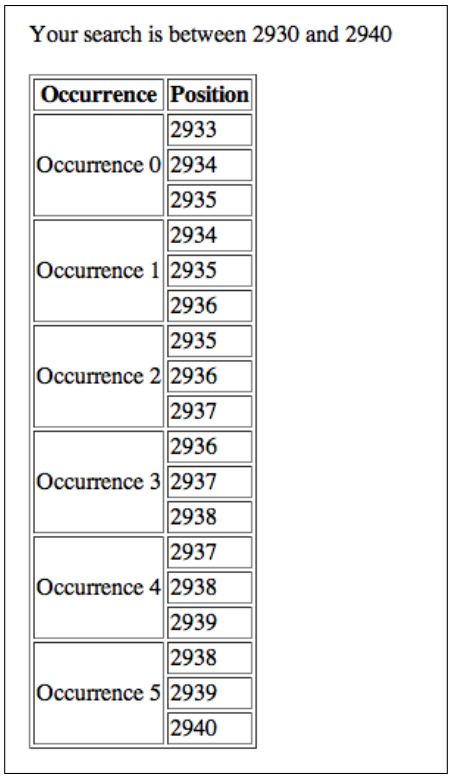


Figure 3.4: Screenshot of third query

pos = vector for storing single matched occurrence
inputs.push_back(mods)
filehandle = open(binary file)
rows = memory_map(filehandle)
// call the binary_search function
num_lines = number of lines in the file
bin_pos = <Binary_Search>(0, num_lines-1, start)
// due to repeated positions in the binary file,
// binary search returns the position of the first match
// move back until you encounter the first occurrence of the position being
//searched
while bin_pos $\geq$ 0 **do**
  if rows[bin_pos]→position $\geq$ start **then**
    decrement bin_pos
  else
    break

```
    end if
  end while

  // counter for number of occurrences of the particular input pattern
  occ_count = 0
  // counter for number of positions found in the range
  found_count = 0
  num_inputs = size of inputs
  while (rows[bin_pos]→position) ≤ end && bin_pos ≤ (num_lines-num_inputs) do
    // store occurrence position since we need to list multiple occurrences
    prev_occ = rows[bin_pos]→position
    for j = 0 to num_inputs do
      mark_string = collect all marks from position //same as rows[bin_pos]→position
      non_inputs = characters other than input marks
      if (mark_string contains inputs[j]) && (mark_string does not contains non_inputs)
      then
        increment found_count
        insert rows[bin_pos]→position into pos
      end if
    end for
    if found_count = num_inputs then
      // all input marks matched in the occurrence
      insert all values from pos into occurrence[occ_count] except between 'r' and 's' insert
      all values from pos into occurrence[occ_count]
      increment occ_count
    end if
    increment bin_pos // until next position value is encountered
  end while
  if occ_count == 0 then
    print "No modifications found"
  else
    display occurrences //display only positions
  end if
end
```

Figure 3.5 shows the result of the fourth query when searched for p-mer 'lh' between the interval 247199170 and 247199190 but not in the interval 247199180 to 247199185. It shows that there are 8 occurrences of this p-mer in the specified interval.

Your search is between 247199170 and 247199190

Occurrence	Position
Occurrence 0	247199175
	247199176
Occurrence 1	247199176
	247199177
Occurrence 2	247199177
	247199178
Occurrence 3	247199178
	247199179
Occurrence 4	247199186
	247199187
Occurrence 5	247199187
	247199188
Occurrence 6	247199188
	247199189
Occurrence 7	247199189
	247199190

Figure 3.5: Screenshot of fourth query

3.7 Search by Modification

Apart from searching by a chromosome, user also has the option to search by a modification. The second query, i.e Find p-mer in (x,y) can be used to search the occurrences of a particular modification in a given range or within the whole chromosome. A p-mer with a single modification, i.e a uni-mer, can be used in such a case.

For example the figure 3.6 shows the result of when a search is done for the modification 'd' (H3K4me1) on chromosome 5 between 60000 and 69000. It shows that there are 14 occurrences of the same. Similarly, a search can be made over the whole chromosome for a particular modification.

Occurrence	Position
Occurrence 0	67828
Occurrence 1	67829
Occurrence 2	67830
Occurrence 3	67831
Occurrence 4	68089
Occurrence 5	68090
Occurrence 6	68091
Occurrence 7	68092
Occurrence 8	68093
Occurrence 9	68094
Occurrence 10	68095
Occurrence 11	68096
Occurrence 12	68097
Occurrence 13	68098
Occurrence 14	68099

Figure 3.6: Search for modification H3K4me1 on chr 5 between 60000 and 69000

3.8 Partitioning Algorithm

A divide and conquer strategy is applied to break down an entire chromosome into three parts recursively, based on the frequency of modifications in each part[6]. A multiple test correction algorithm[7] is applied to reject parts of the chromosome that are significantly similar to each other.

Multiple Test Correction is a 2-pass algorithm. The first pass is essentially a simple chi-square test applied over the entire chromosome and the chromosome is broken down based on the chi-square being significant at the 0.05 level. The probability values are then sorted in ascending order and the following formula [Eq. 1] is applied over each of the probabilities to determine a cut-off probability value. The second pass of the multiple test correction consists of using this cut-off probability value as a threshold to reject further partitioning which denotes that the parts in the chromosome are closely similar in their composition.

Let P_1, P_2, ... P_m be the significant probabilities of the nodes sorted in ascending order.
Let 'k' be the largest 'i' for which $P_{(i)} \leq \frac{i}{m}(0.05)$ *Eq. 1*
Then, we need to re-partition the nodes with probabilities P_1, P_2, ... P_k and terminate partitioning for nodes with probabilities P_{k+1}, P_{k+2}, ... P_m.

If the three parts of the chromosome involved in the multiple test correction differ significantly, then the partition is enacted. If the 3 segments at any level of recursion have insignificant difference in composition, then the recursion is terminated. If the three segments are significantly different, then the segment is again broken down into three more segments and the process is repeated all over again. The result is a ternary tree.

The significance test is carried out as follows:
In any given segment, the positions and modifications are extracted using the first query. An additional piece of code has been used to count the number of modifications of each kind in that segment. These counts are then stored in the 1st row of a 2D array which will hold similar counts from the other two segments. Thus finally we create a 3 X 23 contingency table.

The homogeneity test is applied to this table. It is used to compute the chi-square statistic and the resulting value is compared against the critical value (0.05) for the significance level in the first pass. In the second pass, the critical value changes to the cut-off probability obtained from the first pass.

The partitioning algorithm used here is shown below.

1. Consider the entire chromosome as a single segment. Let the significant probability value, P_{sig} be 0.05.

2. Divide the segment into 3 subsegments.

3. For each subsegment, count the number of each type of modifications and create a row array of counts.

4. Create the 3 X 23 contingency table from the 3 segments.

5. Calculate the chi-square value for the table, and find the chi-square probability value, given degrees of freedom is 44.

6. If the chi-square probability value is $\leq P_{sig}$, goto step 2, else stop the recursion.

7. List all the probabilities in ascending order.

8. Calculate a new probability as specified in Eq. 1

9. Now, let the new probability be assigned to P_{sig}, making it the new cut-off critical value for significance.

10. Start from Step 1 and run the recursive steps 2-6 with the new P_{sig} value.

11. When the chi-square probability value is $>$ the new P_{sig}, stop the recursion and end the program.

3.9 Graphical representation

A tree-like representation is created which allows users to view all the homogeneous intervals within a chromosome. The partition algorithm mentioned in section 3.7 resulted in a ternary tree which is plotted using gnuplot [2]. Twenty four such trees are drawn, one for each chromosome. All these graphs can be viewed in Appendix E or on the website http://giles.genetics.uga.edu/~ankit/. We decided to represent each interval on the graph using its mid-point. The graphs are plotted with locations on the chromosome forming the 'x' co-ordinates and the levels in the tree forming the 'y' co-ordinates for the graph. The

highest level in the graph shows all the homogeneous regions in the chromosome.

3.10 Web Process

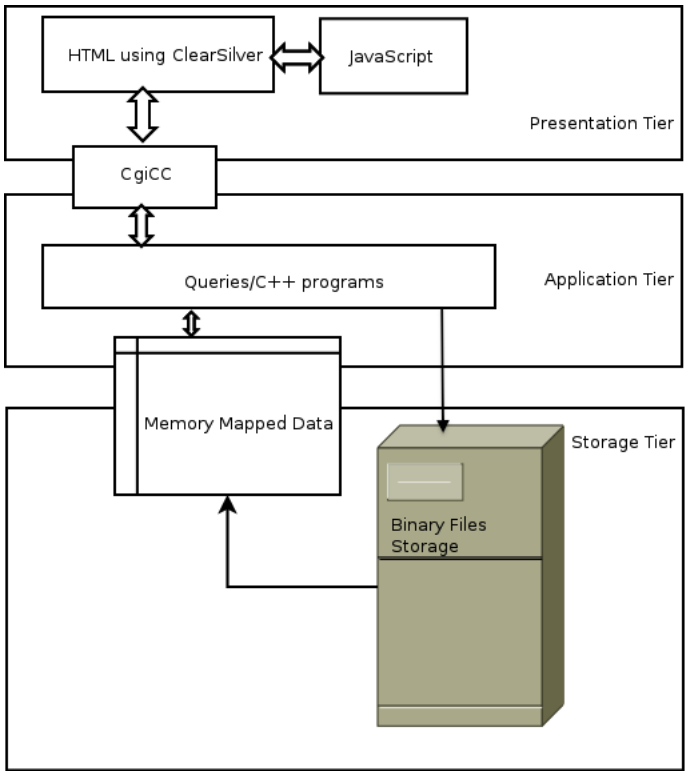


Figure 3.7: Architecture of the Web Application

The web application can be accessed at http://giles.genetics.uga.edu/~ankit/. The application was created as a 3-tier architecture as shown in Figure 3.6. The presentation, application and the data storage have been separated into independent modules whose software and hardware specifications are described on section 2.2 and 2.3 respectively.

Chapter 4

Results

4.1 Query timings

	query1	**query2**	**query3**	**query4**
small (100 - 10000)	0.0048	0.0076	0.0058	0.0081
medium (10000 - 1000000)	1.0648	1.2824	2.1672	1.8256
large (1000000 - 100000000)	8.8345	12.6754	16.4024	13.5689

Figure 4.1: Timing comparison for different range of search intervals in seconds

Figure 4.1 shows the execution times, in seconds, for all the queries over small, medium and large intervals of search ranges. As shown in the table, small range is of interval (100-10000) means that the range of search can vary between 100 to 10000. Ex: A user can search either 100 to 500 (interval of 400) or he can search 350 - 10300 (interval of 9950). Similarly, a medium range is of interval (10000 - 1000000) and a large range is of interval (1000000 - 100000000).

The point to note here is that even though the queries run on huge search intervals, they run very fast and deliver results within seconds. This is largely a result of the memory mapping technique being employed in the queries. A more important benefit of using memory

mapping can be seen in figure 1.2.

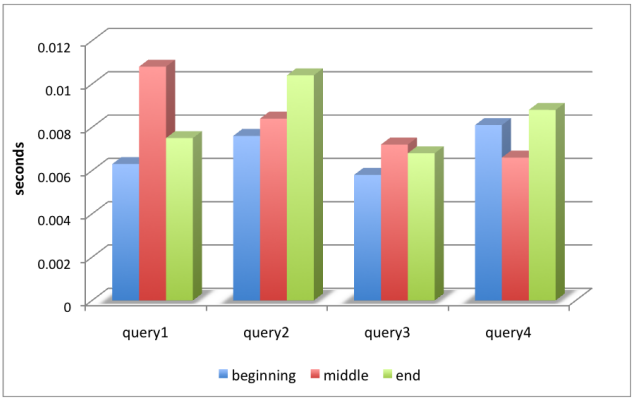


Figure 4.2: Execution time comparison in seconds

Figure 4.2 shows a comparison of the timings when queries are executed at the beginning of the file, in the middle of the file and at the end of the file. As seen, the timings are very similar with only a few milliseconds of difference.

4.2 Coverage

Coverage is defined as the number of times a site was covered by n number of reads of a particular modification. Appendix A gives an overview on the coverages. As seen from the examples in figure 4.3, the coverage has a roughly exponential distribution.

4.3 Run lengths

A location can be defined as a position in nucleotides along a chromosome. A run is defined as a contiguous sequence of locations which consists of DNA reads. Run length is defined as the distance between start and end positions of these runs. There were cases when another run would start before the earlier run has ended. Such runs are known as overlapping runs and in this thesis, they are considered to be different.

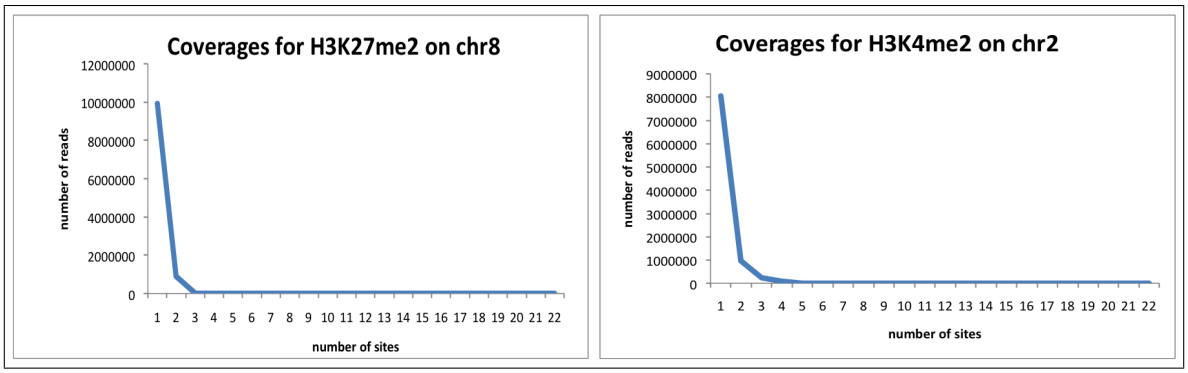


Figure 4.3: Examples of Coverage

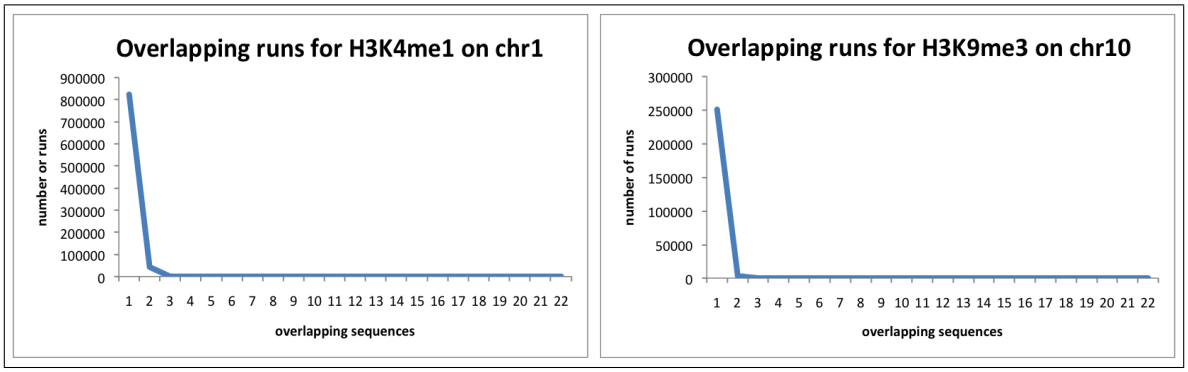


Figure 4.4: Examples of Run Lengths

If such overlapping sequences are considered to be the part of the same run, we would get an exponential curve as shown in figure 4.4 and in Appendix B.

4.4 Spacing

A space is a location with no DNA sequence reads. A run for each kind of modification will have a corresponding space which comprises of empty locations. Every space can be associated with a unique run. Similar to run lengths, even spacing has a exponential curve as shown in figure 4.5

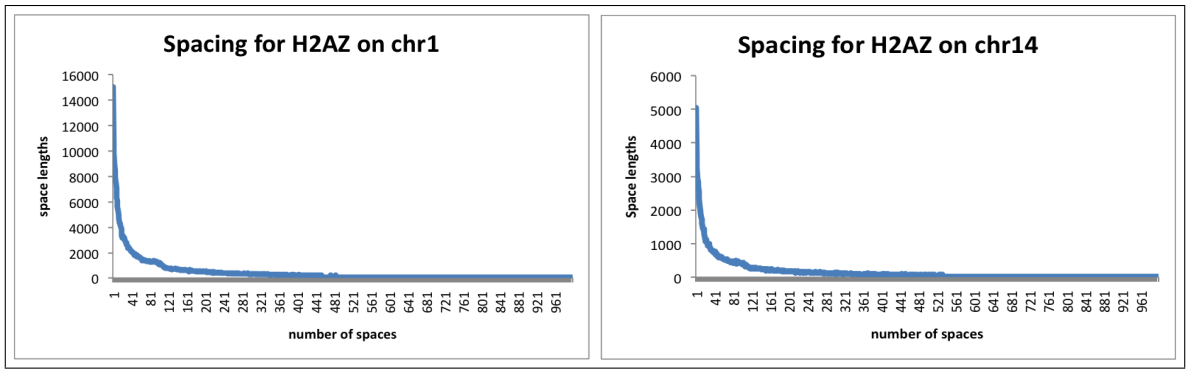


Figure 4.5: Examples of Spacing

4.5 Correlation Coefficient

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

Figure 4.6: Correlation Coefficient formula

A correlation between the run and the spaces was calculated by using the formula as shown in figure 4.6 where *'x'* denotes the run and *'y'* denotes the space.

The correlation coefficient, r , gave very low values as shown in Appendix C, which meant that the runs and spaces are nearly independent of each other.

As seen from Appendix C, many of the correlation coefficient values were negative, which indicates a negative correlation between the runs and spaces. This negative correlation is caused due to the fact that in many cases, the run lengths are longer than their respective spaces.

Chromosome	Total # of modifications
chr1	303228071
chr2	312668742
chr3	250020056
chr4	208347035
chr5	218295125
chr6	212288689
chr7	190666723
chr8	179008125
chr9	145778656
chr10	174430721
chr11	175351822
chr12	171317202
chr13	115187991
chr14	119236849
chr15	109002079
chr16	115784070
chr17	121161507
chr18	92984182
chr19	95937678
chr20	90337142
chr21	45780610
chr22	56506289
chrX	101128973
chrY	10165326

Figure 4.7: Total number of modifications per chromosome

4.6 Total Modifications

Figure 4.7 shows the total number of modifications on every chromosome. These numbers do not count the overlapping sequences, as overlapping runs were considered to be different.

4.7 Multiple Test Correction

Using a multiple test correction reduced the number of terminal nodes by an average of 8%. A table showing this is given in appendix D. The table shows the chromosome number and the terminal nodes obtained before and after using the multiple test correction.

4.8 Graphs

The graphs are drawn with a tree-like representation with x-axis showing the locations on the chromosome and y-axis showing the levels in the tree. We see that there are a lot of non-uniform intervals. Most of the chromosomes get divided up to level 15 (i.e 3^{15} nodes in the tree) which suggest inhomogeneity with large number of intervals with respect to the modifications present within them. Most of the graphs have a big gap, which is likely to be the centromere.

Figure 5 in appendix E shows the graph drawn for chromosome 1. It shows all the homogeneous regions for that chromosome. Each region is denoted by a point representing the center of the homogenous interval. This graph, similar to all others, has been drawn using the partitioning algorithm as described in section 3.8 . The x-axis shows the positions over the 1st chromosome from 0 to 250,000,000. The y-axis represents the levels of the tree, from 1 to 15. This denotes how deep the recursion went through. In this case, it went upto 15 levels. However, not all the homogenous regions are at level 15. Some of them have uniform intervals at lower levels too.

As seen from the graph, there is just one point on the graph at level 1, which represents the first mid-point in the chromosome. At level 2, there are 3 points on the graph and so on until level 15. However, due to lack of proper zooming, it is not possible to see distinct points on the graph at higher levels. So, it seems to be a straight line. There is also a huge gap or space in the graph where there are no points plotted on the graph. This represents the centromere of the chromosome. Hence, we do not see any points on the chromosome in that interval.

Chapter 5

Discussion

The main development here was to create a partition of each chromosome that was locally multinomial. The number of intervals in the partition varied from 200,000 - 6,500,000 within the chromosome. There was thus substantial spatial inhomogeneity in the marks. The graphs also show that the centromeres are apparent.

We have demonstrated some features of the distribution of epigenetic marks. The spacing and the run lengths have a roughly exponential distribution and appear uncorrelated. These observations provides some evidence of the independence in occurrences of the modifications. What remains to be specified is, 'what is determining the variation in probabilities of a modification along a chromosome'.

Another observation which can be noted from the coverages table in Appendix A is that, four modifications namely H3K36me1, H3K36me3, H3K79me1 and H3K79me2, have a peculiar distribution. But surprisingly the runs and spacing of these modifications is quite similar to the rest of the modifications. It would be interesting to find out the reason behind this. We decided to include these modifications in our further calculations as leaving them out would have results in different counts for our study with only 19 modifications.

We have created the partition for all the chromosomes in the human genome. It would

be desirable to identify characteristic motifs within elements of the partition and relate the elements of the partition by composition [9].

Here we have considered overlapping sequences of the same kind of modification as separate runs. We might have considered these overlapping sequences as a part of the same run, which would have eventually led us with comparatively less number of counts.

Before using the multiple test correction, Fishers exact test[3] along with chi-square test was used for the partition algorithm. But this process increased the number of terminal nodes by around 20-30%. A reason for this might be the size of the tables. The table size used here is 3 X 23 (i.e 69 entries). This might be a huge table for the Fishers exact test to calculate on due to which we might not get appropriate results. The reason for this can be researched in the future.

The partition algorithm used here results in a large number of terminal nodes. We might reduce the number of these terminal nodes, by say clustering on composition.

Another limitation of our procedure is optimizing the partition. Different approaches could have been used to create a better partition in some sense. For example, some kind of model averaging method could have been used to create the partition boundaries [6].

Chapter 6

Conclusion

Histones in a human nucleosome are subject to various post-translation modifications such as acetylation, methylation, phosporylation, etc. The modifications play a important role in gene regulation. The kind of modification and it location in a DNA sequence is called epigenetic code. The Epigenetic code is responsible in part for developing and maintaining a human body structure. This thesis deals with creating a web application for users to quickly find the epigenetic code along the human chromosomes.This thesis aims at finding a local structure of the histone modifications in a human genome.

To ascertain this, we created a data structure that can allow users to view quickly the spatial local locations of the histone modifications in a DNA sequence. The information about the histone modifications, i.e type of the modification, its location over the chromosome and their number of reads, were there stored in this data structure format which was then written in a binary files and stored on the server. A memory mapping technique and a binary search algorithm were used to search for a specific location in theses binary files. Over results show that this process very less time and works very efficiently.

Another aim of this thesis is to find all the homogeneous regions in the human chromosome. For this, we used a divide and conquer strategy with the partition algorithm as

mentioned in 3.7. This resulted in a ternary tree which is represented with the chromosome positions on the x-axis and the levels in the trees on the y-axis. The graphs show that there are a large number of terminal nodes in the trees, which represent large non-uniformity in these regions. The graphs show that almost all the chromosomes get divided up to 15 levels and hence we infer that there are a large number of homogeneous regions in the partition of each chromosome.

Bibliography

[1] Artem Barski, Suresh Cuddapah, Kairong Cui, Tae-Young Roh, Dustin E. Schones, Zhibin Wang, Gang Wei, Iouri Chepelev, Keji Zhao. High-Resolution Profiling of Histone Methylations in the Human Genome. *Cell* - 18 May 2007 (Vol. 129, Issue 4, pp. 823-837)

[2] gnuplot
http://www.gnuplot.info/

[3] C. R. Mehta and N. R. Patel, Fexact−a fortran subroutine for fisher exact test on unordered RxC contingency-tables, ACM Transactions on Mathematical Software, vol. 12, no. 2, pp. 154161, 1986.

[4] David R. Bentley et. al. Accurate whole human genome sequencing using reversible terminotor chemistry. *Nature* - 6 Nov 2008; doi:10.1038/nature07517.

[5] High-Resolution Profiling of Histone Methylations in the Human Genome. LMI, DIR, NHLBI, NIH
http://dir.nhlbi.nih.gov/papers/lmi/epigenomes/hgtcell.aspx

[6] Zhang Z, Townsend JP, 2009 Maximum-Likelihood Model Averaging To Profile Clustering of Site Types across Discrete Linear Sequences. *PLoS Comput Biol* 5(6): e1000421. doi:10.1371/journal.pcbi.1000421

[7] Yoav Benjamini, Yosef Hochberg, "Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing". *Journal of the Royal Statistical Society. Series B (Methodological)*-1995, (Vol. 57, pp. 289-300)

[8] Srinivas Aluru. *Handbook of Computational Molecular Biology*, Chapter 37. Chapman & Hall/CRC,2006

[9] Hon G, Ren B, Wang W, 2008 ChromaSig: A Probabilistic Approach to Finding Common Chromatin Signatures in the Human Genome. *PLoS Computational Biology* 4(10): e1000201. doi:10.1371/journal.pcbi.1000201

Appendix A

	H2AZ	H2BK5me1	H3K27me1	H3K27me2	H3K27me3	H3K36me1	H3K36me3	H3K4me1	H3K4me2	H3K4me3	H3K79me1	H3K79me2
1	109798608	116993291	190031557	175263995	169747591	0	7	125936644	91931056	176775507	0	0
2	18220632	25226099	27334283	20261093	20672688	172207966	209642323	30179482	14391065	24425468	0	0
3	10168001	6698090	3093975	1369626	1990376	0	15	9615427	2910658	4859941	0	0
4	2043213	3479007	1197498	582363	585721	12547234	35735190	5645413	1479142	3279172	0	0
5	818443	1351745	101574	27854	42241	0	0	2662477	468689	1869579	1327	1378
6	1464953	937199	126539	48737	47426	914047	8891510	1961743	328746	1582042	0	21
7	314500	427052	9164	7074	4762	0	0	1026748	108783	1125540	93944079	59716032
8	266629	341506	22342	9069	7394	88398	2995308	845745	95526	1004073	0	0
9	351502	168198	3898	4014	2383	0	0	448014	30916	756765	0	0
10	136647	150503	5871	3136	2637	18641	1230044	407018	35036	696497	155	70
11	77791	79926	2401	2334	1577	0	0	213218	9473	537451	0	0
12	161531	76504	2622	2024	1204	9691	577368	212886	13492	505696	0	49
13	45385	41686	1706	1705	1154	0	0	110743	3278	393998	0	12
14	47337	41788	1458	1381	1018	5953	302123	120193	6225	377368	14376878	13440301
15	71931	24588	1375	1175	913	0	0	58765	1445	294691	9	1
16	30332	25027	1233	1110	805	4848	173597	70324	3142	287721	0	0
17	18142	15981	1081	1067	772	0	0	33586	643	227229	0	17
18	47087	15759	814	701	566	3978	106611	43837	1563	226310	0	0
19	12165	10488	765	867	429	0	0	20405	308	178247	0	0
20	14240	11093	648	769	528	3188	69494	27737	744	179398	0	2
21	24827	7493	645	636	349	0	0	12344	179	141453	2695724	4337701
22	9944	8036	676	546	529	2450	49410	18290	532	144158	0	0

	H3K79me3	H3K9me1	H3K9me2	H3K9me3	H3R2me1	H3R2me2	H4K20me1	H4K20me3	H4R3me2	Pol2	CTCF
1	53220659	147058494	194294632	106913989	180325736	124673775	105721161	76269644	152132578	66827464	43587512
2	27171216	28180301	24540087	14177001	23628454	17081068	27619464	10690274	9549359	9965219	5976906
3	6411826	6010807	1993409	1891081	2022315	1799057	9575282	1918088	710353	2207077	932383
4	3488578	2821259	722908	747012	885114	701061	5636880	959172	124035	753168	560259
5	1800198	732347	45534	165130	62132	77653	2773023	409997	58874	346398	276033
6	1077778	512976	62298	123825	87501	85200	2014767	316504	38179	187430	241181
7	642327	122393	9562	39348	10478	17058	1120829	192958	26733	110694	165703
8	401362	128037	12307	37796	15601	18774	909169	166722	20884	70090	148421
9	251940	24810	4915	16304	5161	7654	537344	116732	15465	45685	107705
10	161247	39354	3821	15808	5442	7659	463959	103937	11837	31229	100300
11	103891	6355	3398	8248	3256	4682	286802	75362	9994	21901	74025
12	68265	14579	2875	7962	2991	4421	257375	69113	8324	15693	70073
13	44488	2292	2150	5346	2156	3185	162704	52769	6398	11526	51758
14	30178	5706	1923	5483	1998	3099	153131	51099	5695	8942	49016
15	20173	1043	1585	3655	1419	2227	97982	39962	4857	6884	35475
16	13779	2590	1388	3245	1663	2277	94270	38452	3990	5481	34635
17	9167	485	1031	2588	1397	2004	61297	28934	3594	4534	24692
18	6388	1137	1188	2722	1201	1729	60334	28179	3391	3884	24308
19	4338	314	806	2039	891	1674	39664	22468	2624	3316	16915
20	3067	657	801	1716	1052	1369	40201	22321	2443	2565	17423
21	2194	212	604	1432	836	1296	26231	17943	2131	2368	11598
22	1686	409	762	1482	759	1123	27375	17866	1781	2027	12004

Figure 1: Table showing the coverages for all the 23 modifications

Appendix B

H2AZ	H2BK5me1	H3K27me1	H3K27me2	H3K27me3	H3K36me1	H3K36me3	H3K4me1	H3K4me2	H3K4me3	H3K79me1	H3K79me2
6449656	7433400	9748956	8990667	8743975	8028907	11988296	9062443	5093850	11365689	4938075	4160877
252144	241991	146110	78471	86375	57452	391464	432424	125221	628115	74048	161022
35118	28711	4335	1648	1601	1295	30792	66507	9576	196763	2070	16645
7573	4896	479	355	247	351	4601	13288	979	84623	82	2454
1811	1381	173	152	80	147	1148	3329	151	35852	11	408
566	419	96	76	55	83	389	951	28	16079	4	118
192	187	61	37	34	47	159	299	12	7522	1	26
74	62	34	32	12	35	102	107	2	3879	2	9
38	36	17	9	12	24	44	56	2	2037	2	1
22	22	17	9	5	16	23	20	1	1128	2	1
7	11	15	16	4	7	22	7	0	648	0	0
8	9	10	8	2	9	11	10	1	336	0	0
4	8	7	5	2	10	13	4	0	233	0	0
1	3	8	2	3	8	6	3	0	135	0	0
2	3	4	0	1	2	5	7	0	99	0	0
1	0	1	1	2	1	0	3	0	50	0	0
1	2	1	1	0	2	6	2	0	32	0	0
0	1	6	2	1	3	5	1	0	23	0	0
0	2	0	1	1	2	4	0	0	11	0	0
0	0	1	3	0	1	2	0	0	8	0	0
0	0	0	0	0	1	1	0	0	11	0	0
0	1	0	0	0	0	1	1	0	2	0	0

H3K79me3	H3K9me1	H3K9me2	H3K9me3	H3R2me1	H3R2me2	H4K20me1	H4K20me3	H4R3me2	Pol2	CTCF
4143848	8534215	9540738	5950082	9239999	6066335	8254038	4027524	7231492	3627523	2421267
159295	250777	123318	87235	132348	36588	557832	145813	45811	123304	73877
17616	17097	3616	6072	3806	1784	99102	28021	3662	9520	18768
2910	1546	519	1224	566	644	25375	10161	1398	2200	6326
596	201	212	418	234	261	8754	4411	649	787	2531
141	51	120	185	143	164	3458	2343	383	364	1186
44	24	74	99	79	97	1593	1477	188	184	470
15	11	39	48	71	49	802	978	150	120	264
5	4	37	56	45	38	424	630	88	100	141
3	5	22	30	22	25	237	379	63	51	81
2	2	15	20	17	19	155	274	41	42	52
1	1	6	18	15	15	86	209	37	22	33
0	1	7	13	15	12	61	159	19	10	22
0	2	4	8	9	5	32	130	25	20	13
0	0	4	4	7	6	20	77	17	17	11
0	1	1	4	3	2	14	59	12	13	8
0	0	5	4	5	6	10	56	7	8	2
0	1	5	5	2	2	9	35	9	13	3
0	0	1	4	2	2	11	34	6	2	4
0	2	1	3	3	2	3	28	7	1	1
0	0	1	3	4	2	6	26	2	6	1
0	0	1	6	1	3	6	24	7	2	1

Figure 2: The total run lengths for all the modifications for all the chromosomes

Appendix C

	chr1	chr2	chr3	chr4	chr5	chr6	chr7	chr8	chr9	chr10	chr11	chr12
H2AZ	0.0078	-0.0132	-0.0080	-0.0113	-0.0118	-0.0126	-0.0117	-0.0102	-0.0020	-0.0120	-0.0116	-0.0227
H2BK5me1	0.0678	-0.0115	-0.0037	-0.0124	-0.0113	-0.0100	-0.0104	-0.0105	-0.0018	-0.0107	-0.0098	-0.0187
H3K27me1	0.0054	0.0037	-0.0162	-0.0032	-0.0032	-0.0036	-0.0028	-0.0028	-0.0077	-0.0040	-0.0030	-0.0063
H3K27me2	0.0145	-0.0016	-0.0026	-0.0019	-0.0016	0.0335	-0.0015	-0.0013	-0.0203	0.0102	-0.0013	-0.0031
H3K27me3	0.0742	-0.0029	-0.0011	-0.0027	-0.0021	0.0327	-0.0023	-0.0022	-0.0096	-0.0028	-0.0025	-0.0045
H3K36me1	0.1792	0.0106	-0.0018	-0.0016	-0.0043	-0.0015	-0.0011	-0.0095	0.0194	-0.0018	-0.0099	-0.0025
H3K36me3	0.1833	-0.0066	-0.0084	-0.0069	-0.0067	-0.0055	-0.0058	-0.0053	-0.0010	-0.0060	-0.0051	-0.0118
H3K4me1	0.0845	-0.0154	-0.0046	-0.0156	-0.0147	-0.0087	-0.0125	-0.0128	-0.0022	-0.0129	-0.0115	-0.0223
H3K4me2	0.0413	-0.0140	-0.0103	-0.0147	-0.0140	-0.0132	-0.0115	-0.0116	-0.0021	-0.0130	-0.0115	-0.0226
H3K4me3	0.0127	-0.0150	-0.0093	-0.0140	-0.0137	-0.0064	-0.0132	-0.0116	-0.0022	-0.0140	-0.0127	-0.0268
H3K79me1	0.0286	0.0122	-0.0102	-0.0121	-0.0104	-0.0087	-0.0092	-0.0096	-0.0017	-0.0096	-0.0084	-0.0154
H3K79me2	-0.0017	-0.0217	-0.0074	-0.0127	-0.0216	-0.0180	-0.0182	-0.0182	-0.0031	-0.0191	-0.0169	-0.0315
H3K79me3	0.0212	-0.0220	-0.0149	-0.0213	-0.0216	-0.0012	-0.0184	-0.0174	-0.0032	-0.0195	-0.0170	-0.0322
H3K9me1	0.0203	-0.0105	-0.0145	-0.0110	-0.0100	-0.0093	-0.0079	-0.0084	-0.0015	-0.0094	-0.0079	-0.0162
H3K9me2	-0.0017	0.0293	-0.0071	0.0079	-0.0012	-0.0016	-0.0015	0.0194	-0.0022	-0.0015	-0.0014	0.0088
H3K9me3	-0.0005	-0.0052	-0.0010	-0.0037	-0.0042	0.0304	-0.0049	-0.0038	-0.0007	-0.0064	-0.0044	-0.0083
H3R2me1	0.1694	-0.0018	-0.0032	-0.0018	-0.0011	-0.0014	-0.0014	-0.0010	-0.0027	-0.0022	0.0101	-0.0023
H3R2me2	0.1413	-0.0023	-0.0012	-0.0018	-0.0079	-0.0015	0.0127	-0.0013	-0.0034	-0.0032	0.0037	-0.0023
H4K20me1	0.0677	-0.0172	-0.0071	-0.0198	-0.0173	-0.0148	-0.0153	-0.0161	-0.0027	-0.0154	-0.0142	-0.0277
H4K20me3	-0.0029	-0.0075	-0.0118	-0.0117	-0.0081	-0.0038	-0.0125	0.0033	-0.0014	-0.0141	-0.0055	-0.0246
H4R3me2	0.0818	-0.0025	-0.0057	0.0384	-0.0012	-0.0012	-0.0029	0.0139	-0.0032	-0.0051	-0.0013	-0.0023
Pol2	0.1532	-0.0118	-0.0039	-0.0080	-0.0103	-0.0036	-0.0107	-0.0083	-0.0018	-0.0107	-0.0104	-0.0194
CTCF	0.0326	-0.0234	-0.0071	-0.0187	-0.0206	-0.0211	-0.0210	-0.0193	-0.0037	-0.0240	-0.0215	-0.0434

	chr13	chr14	chr15	chr16	chr17	chr18	chr19	chr20	chr21	chr22	chrX	chrY
H2AZ	-0.0445	-0.0728	-0.0421	-0.0035	-0.0556	-0.0153	0.0020	-0.0116	-0.0060	-0.0608	-0.0131	-0.0019
H2BK5me1	-0.0454	-0.0574	-0.0352	-0.0025	-0.0353	-0.0153	-0.0023	0.0108	-0.0057	-0.0334	-0.0101	-0.0020
H3K27me1	-0.0140	-0.0286	-0.0170	-0.0082	-0.0128	-0.0047	-0.0010	-0.0038	-0.0017	-0.0171	-0.0025	-0.0014
H3K27me2	-0.0067	-0.0133	-0.0064	-0.0039	-0.0084	-0.0026	-0.0084	-0.0015	-0.0013	-0.0030	-0.0011	-0.0015
H3K27me3	-0.0110	-0.0194	-0.0090	-0.0008	-0.0144	-0.0027	-0.0098	-0.0022	-0.0011	-0.0088	-0.0022	-0.0011
H3K36me1	-0.0031	-0.0073	-0.0058	-0.0047	0.0005	-0.0023	-0.0009	0.2497	-0.0012	-0.0053	-0.0017	-0.0011
H3K36me3	-0.0268	-0.0462	-0.0295	-0.0014	-0.0241	-0.0083	-0.0026	-0.0080	0.0092	-0.0247	-0.0062	-0.0012
H3K4me1	-0.0561	-0.0705	-0.0472	-0.0029	-0.0419	-0.0201	-0.0025	0.0201	-0.0066	-0.0412	-0.0136	-0.0018
H3K4me2	-0.0465	-0.0537	-0.0384	-0.0024	-0.0393	-0.0186	-0.0028	0.0407	-0.0063	-0.0337	-0.0131	-0.0023
H3K4me3	-0.0606	-0.1151	-0.0619	-0.0034	-0.0705	-0.0174	-0.0036	0.0358	-0.0062	-0.0703	-0.0174	-0.0016
H3K79me1	-0.0018	-0.0309	-0.0013	-0.0019	-0.0188	-0.0158	-0.0017	0.0085	0.0081	-0.0010	-0.0052	-0.0018
H3K79me2	-0.0034	-0.0582	-0.0028	-0.0043	-0.0397	-0.0307	0.0191	-0.0157	0.0137	-0.0022	-0.0186	-0.0031
H3K79me3	-0.0697	-0.0711	-0.0459	-0.0043	-0.0495	-0.0286	-0.0034	0.0379	-0.0093	-0.0487	-0.0194	-0.0028
H3K9me1	-0.0406	-0.0530	-0.0312	-0.0021	-0.0305	-0.0136	-0.0021	-0.0076	-0.0047	-0.0299	-0.0092	-0.0018
H3K9me2	-0.0039	-0.0105	-0.0068	-0.0084	-0.0150	-0.0024	-0.0010	-0.0024	-0.0015	-0.0029	0.0164	-0.0014
H3K9me3	-0.0120	-0.0246	-0.0226	-0.0019	-0.0241	-0.0050	-0.0029	-0.0058	-0.0032	-0.0366	-0.0036	-0.0017
H3R2me1	-0.0061	-0.0086	-0.0052	-0.0005	-0.0046	-0.0027	-0.0051	0.1669	0.0188	-0.0071	-0.0092	-0.0028
H3R2me2	-0.0020	-0.0041	-0.0035	-0.0011	-0.0089	-0.0019	-0.0068	-0.0022	-0.0020	-0.0095	0.0208	-0.0024
H4K20me1	-0.0622	-0.0676	-0.0468	-0.0033	-0.0447	-0.0251	-0.0029	-0.0126	0.0154	-0.0407	-0.0192	-0.0024
H4K20me3	-0.0144	-0.0240	-0.0495	-0.0031	-0.0229	-0.0098	-0.0037	-0.0071	-0.0106	-0.0491	-0.0120	-0.0028
H4R3me2	-0.0025	-0.0071	-0.0021	-0.0012	-0.0080	-0.0024	-0.0072	0.0066	-0.0031	-0.0112	-0.0028	-0.0021
Pol2	-0.0340	-0.0540	-0.0297	-0.0025	-0.0314	-0.0135	-0.0027	-0.0094	-0.0053	-0.0338	-0.0121	-0.0023
CTCF	-0.0680	-0.1036	-0.0687	-0.0065	-0.0874	-0.0279	-0.0072	0.0295	-0.0105	-0.0876	-0.0218	-0.0030

Figure 3: Table showing correlation coefficient of spacing and run lengths

Appendix D

Chromosome #	Terminal nodes before multiple test correction	Terminal nodes after multiple test correction
1	6985471	6501661
2	7386077	6912765
3	5652315	5149539
4	4579833	4374105
5	4700193	4421525
6	4509073	4281157
7	4135103	4009053
8	4037995	3923059
9	3928107	3852265
10	3854737	3736259
11	3865791	3747787
12	3819251	3698983
13	2862635	2705747
14	2780659	2626505
15	2507371	2365689
16	2534539	2394211
17	2577619	2437795
18	2258731	2031463
19	1890361	1757899
20	1988487	1988487
21	1057471	1008587
22	1135587	1090799
X	2920127	2707119
Y	262679	229685

Figure 4: Number of terminal nodes before and after applying the multiple test correction

Appendix E

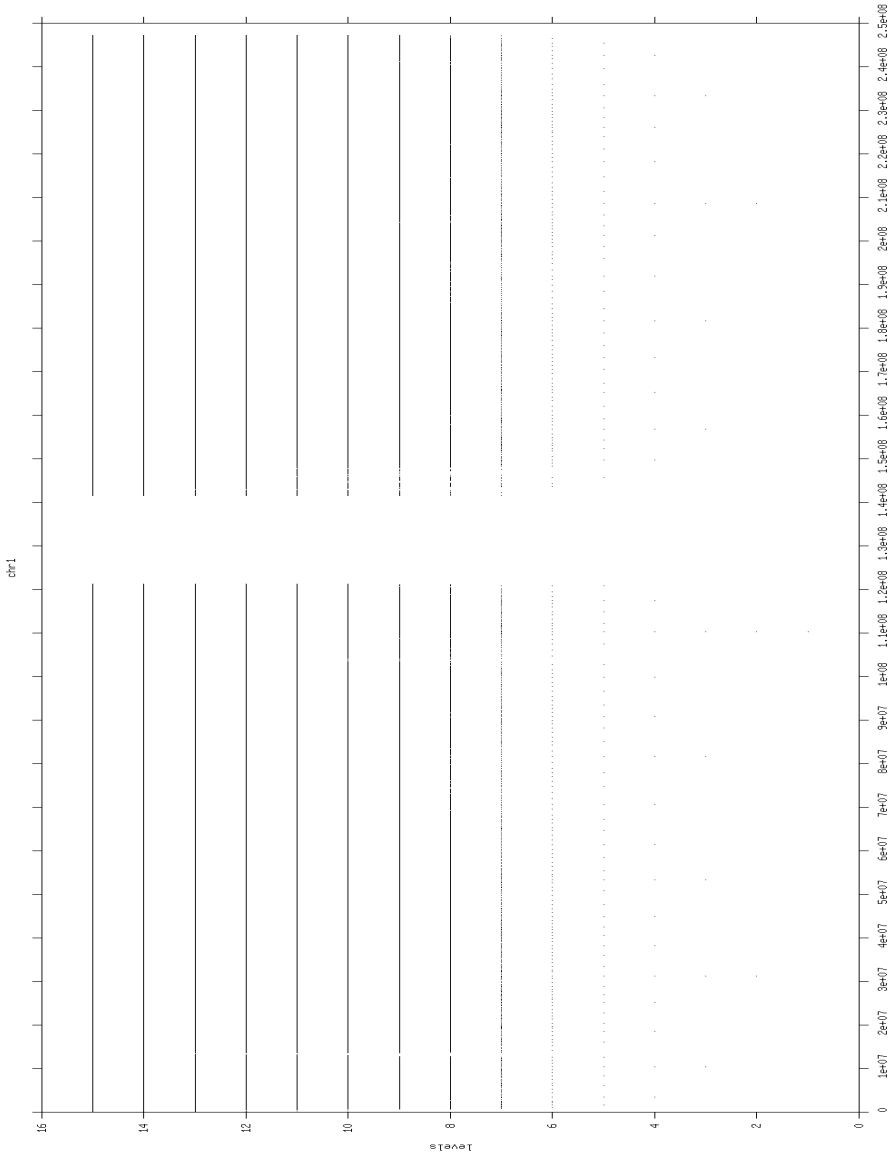


Figure 5: graphical representation of the partitions on chromosome 1

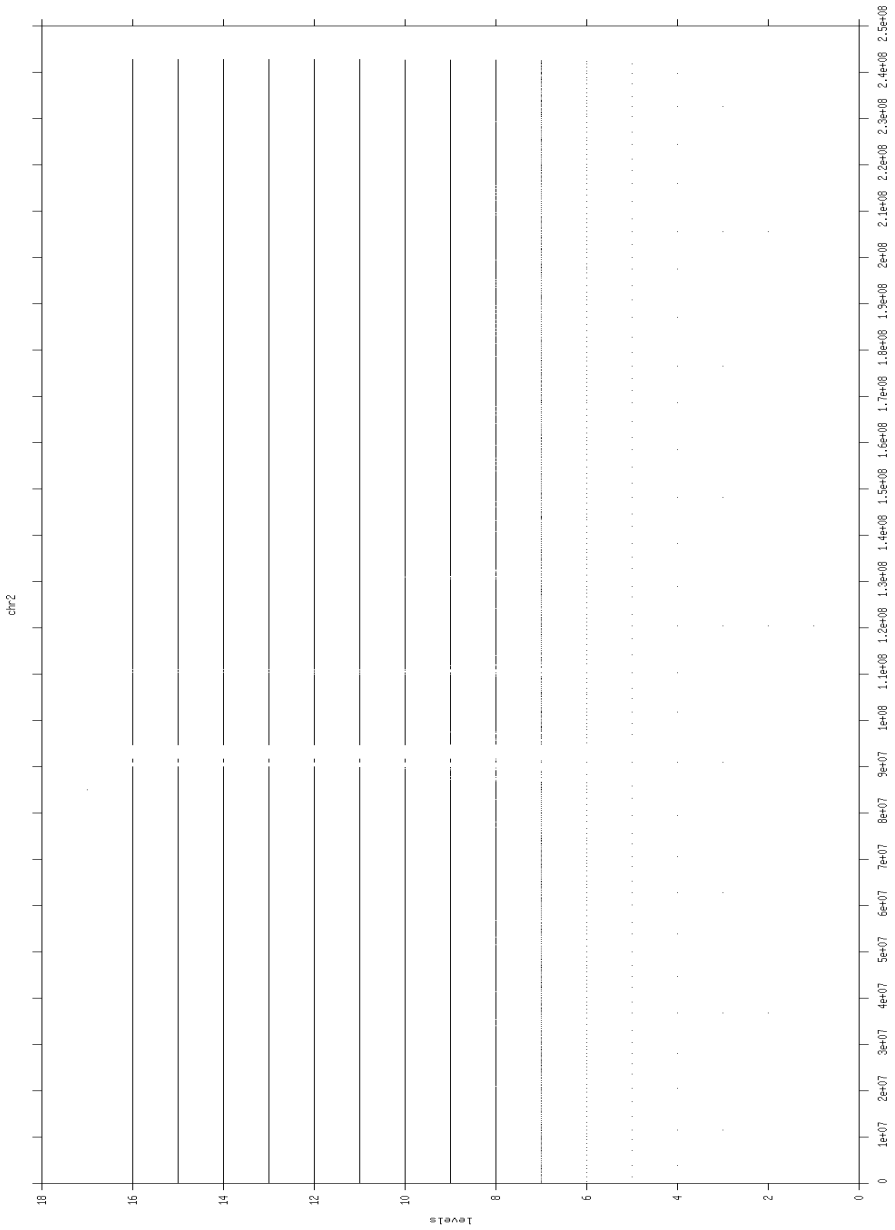


Figure 6: graphical representation of the partitions on chromosome 2

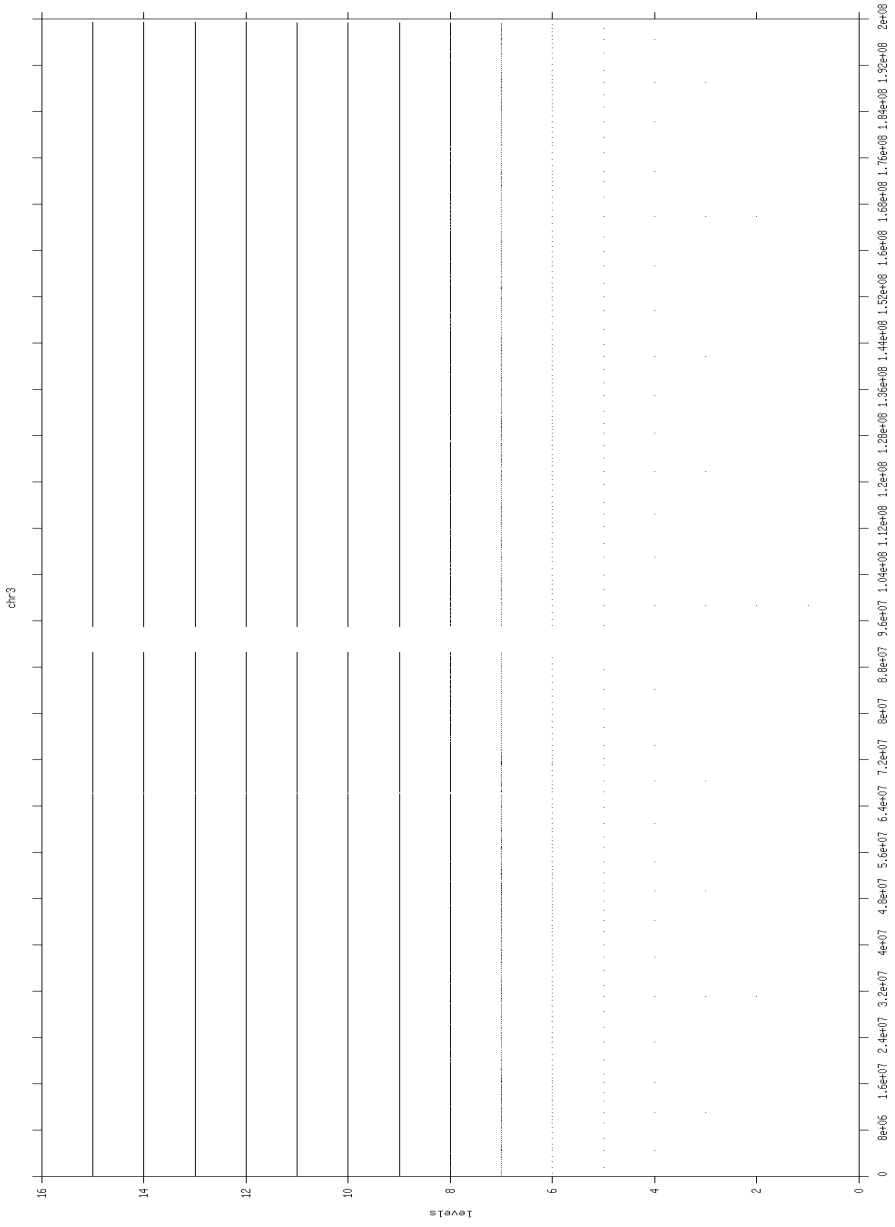


Figure 7: graphical representation of the partitions on chromosome 3

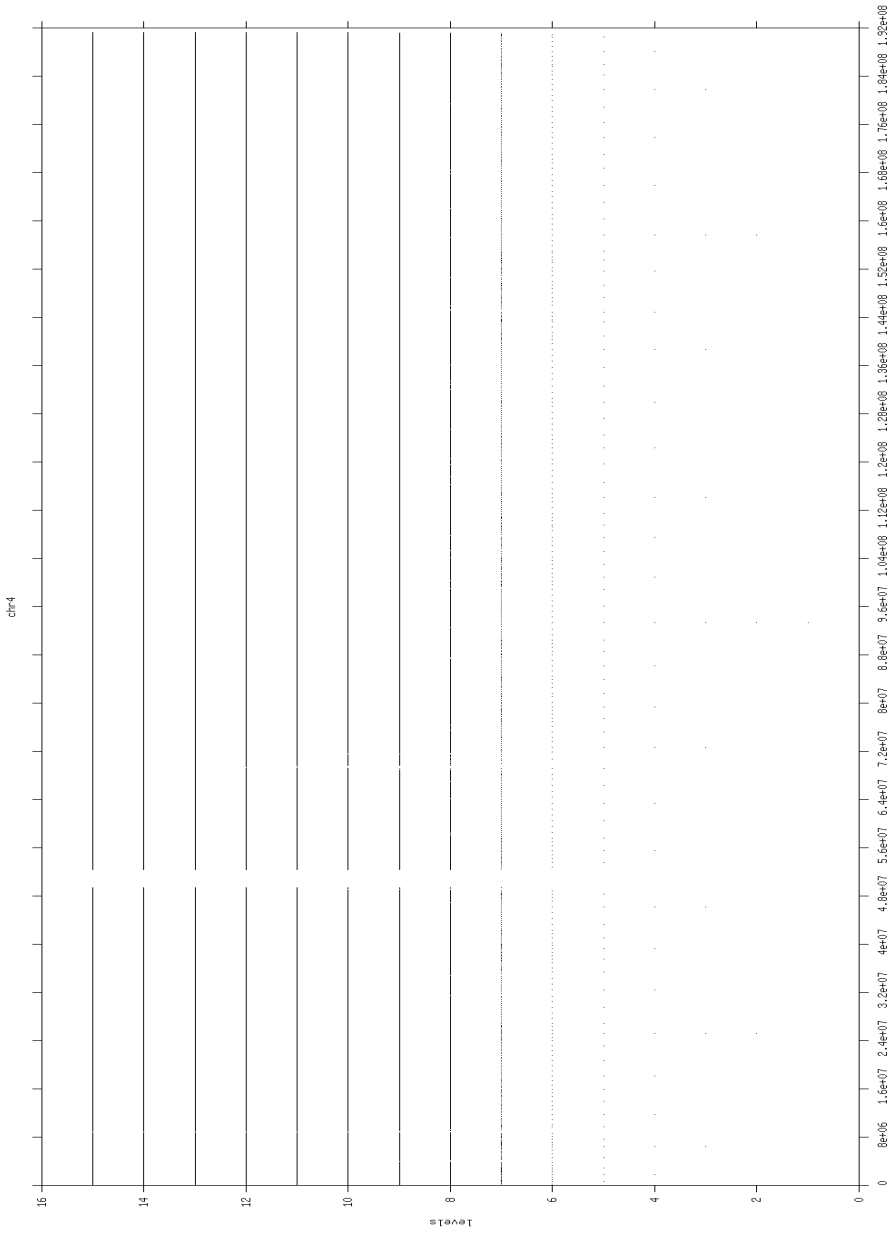


Figure 8: graphical representation of the partitions on chromosome 4

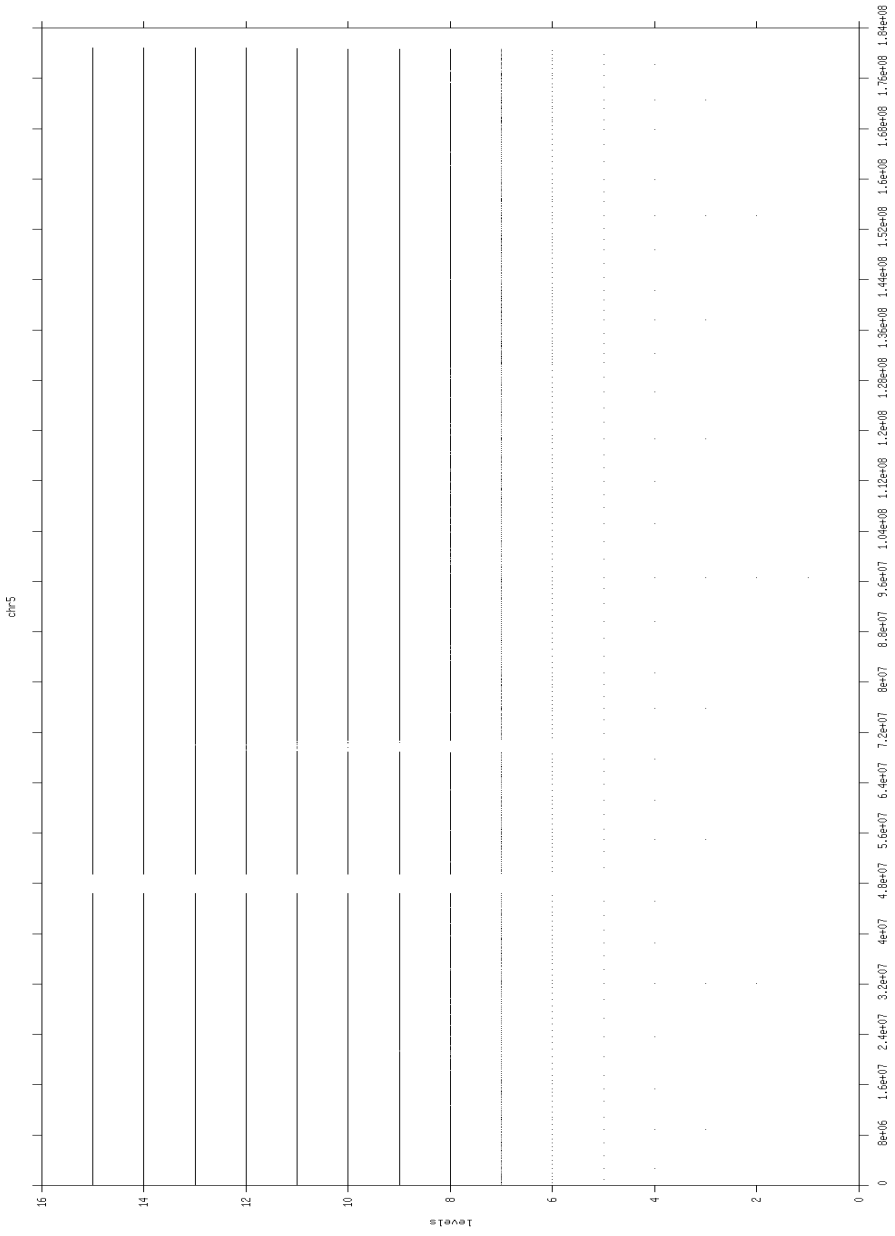


Figure 9: graphical representation of the partitions on chromosome 5

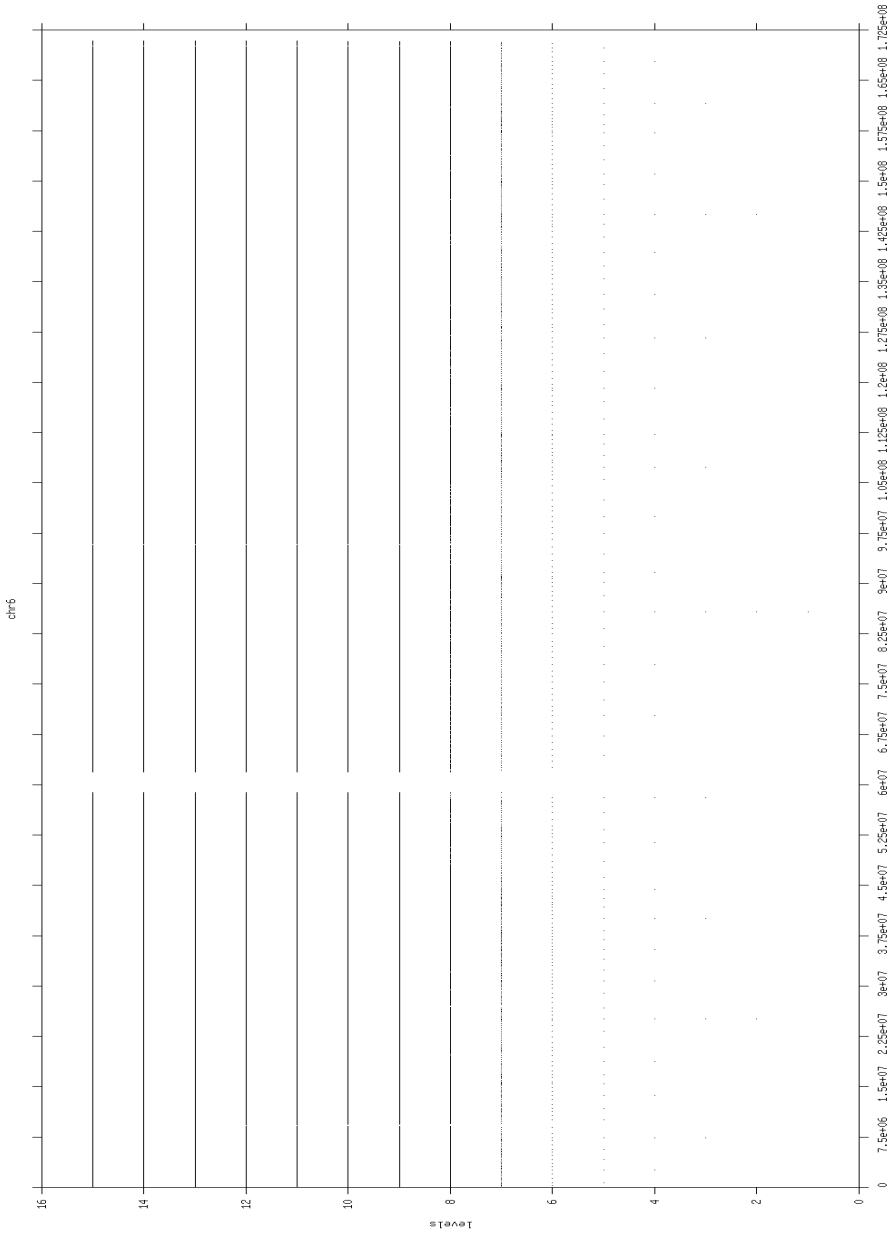


Figure 10: graphical representation of the partitions on chromosome 6

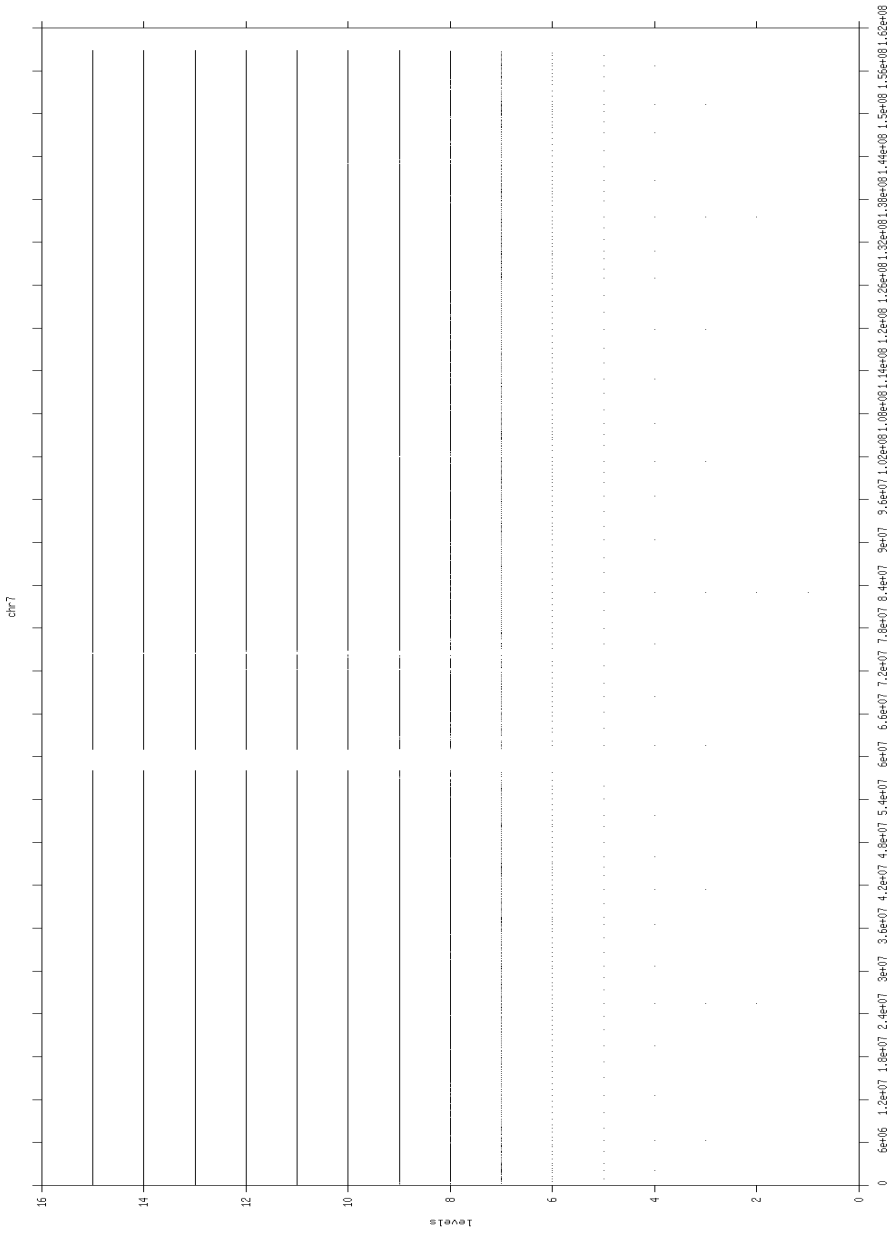


Figure 11: graphical representation of the partitions on chromosome 7

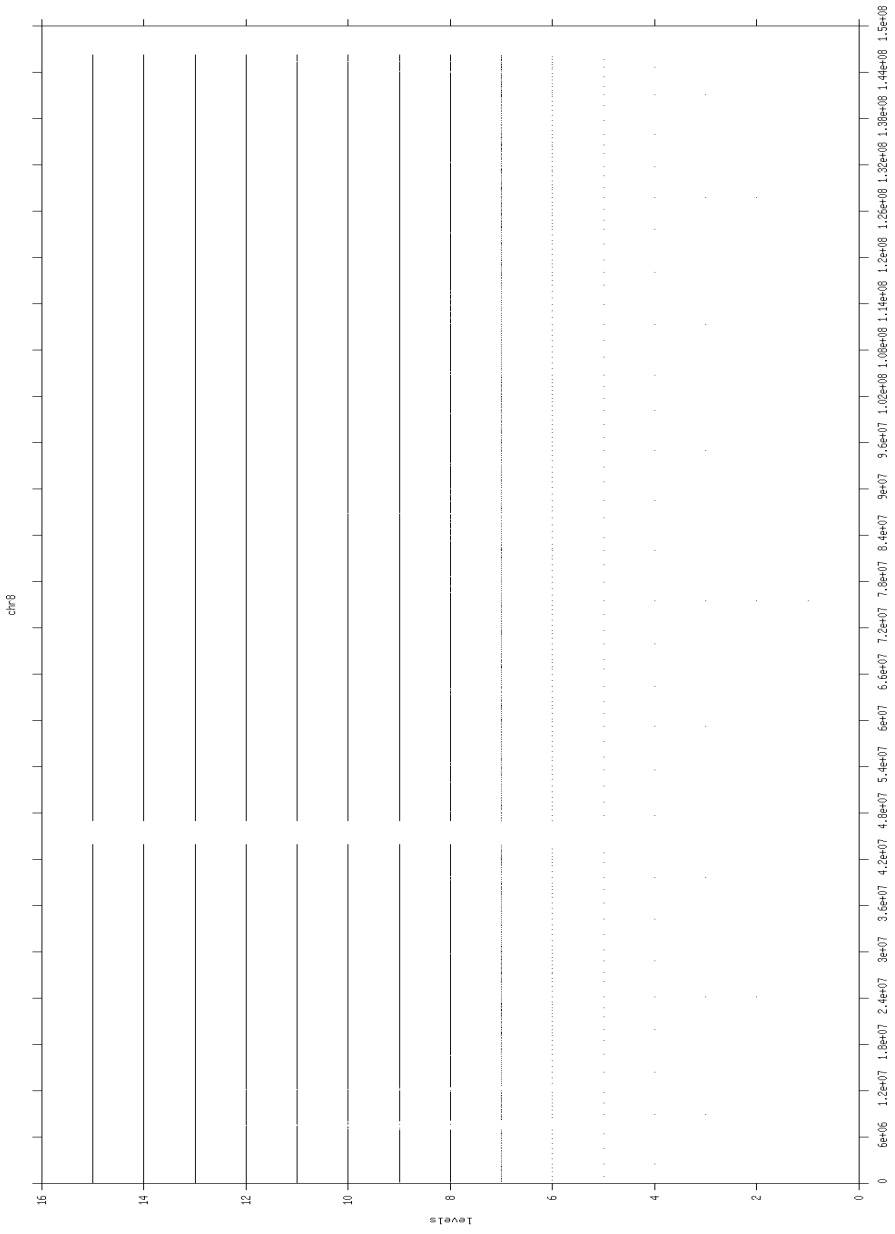


Figure 12: graphical representation of the partitions on chromosome 8

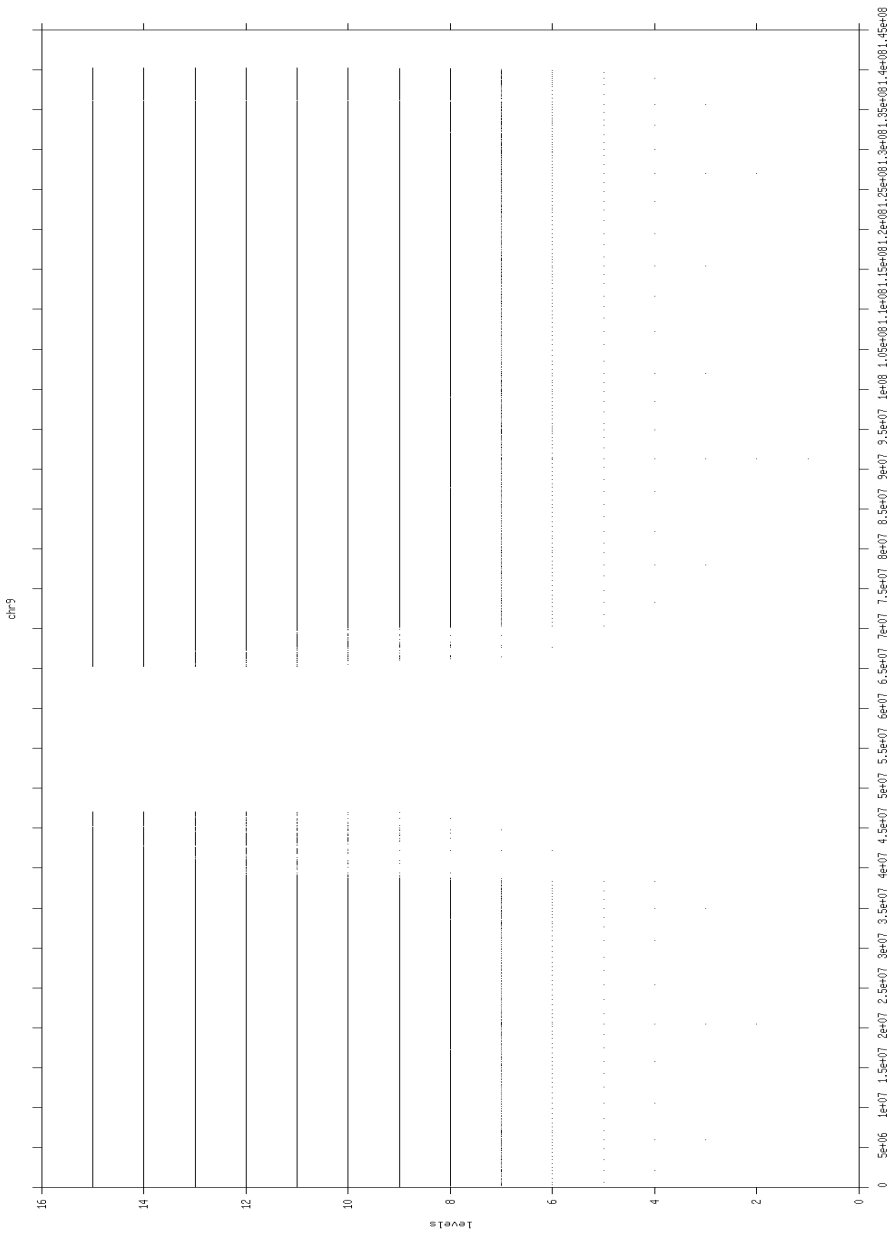


Figure 13: graphical representation of the partitions on chromosome 9

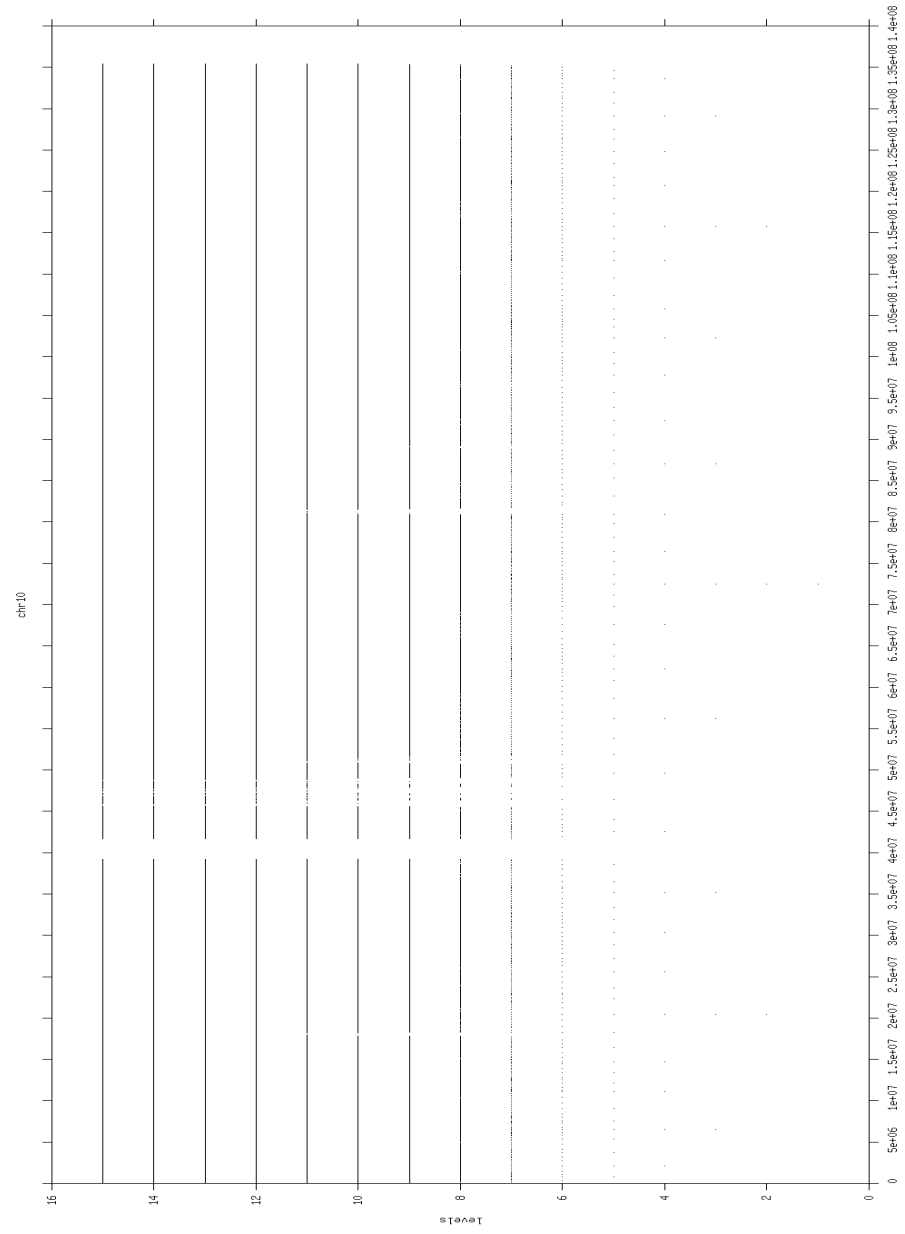


Figure 14: graphical representation of the partitions on chromosome 10

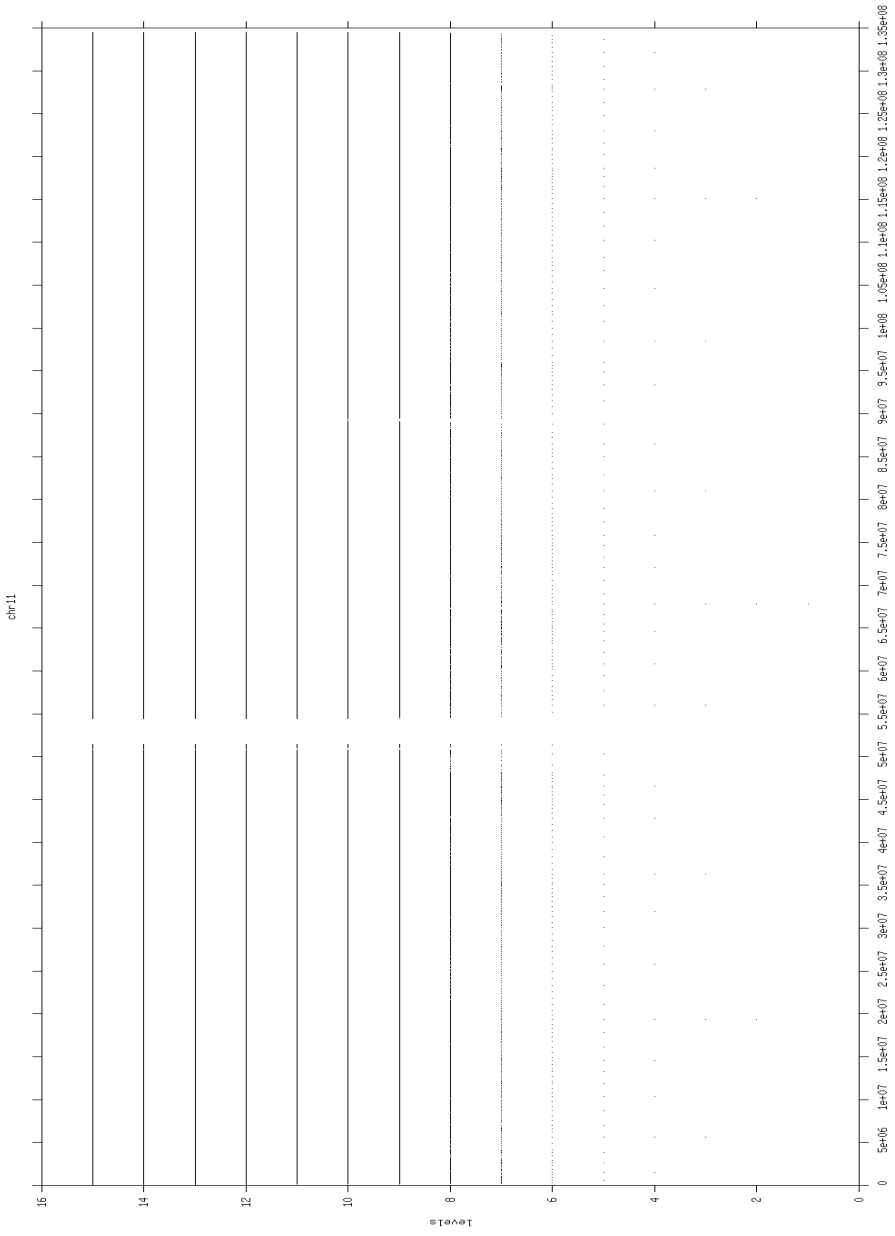


Figure 15: graphical representation of the partitions on chromosome 11

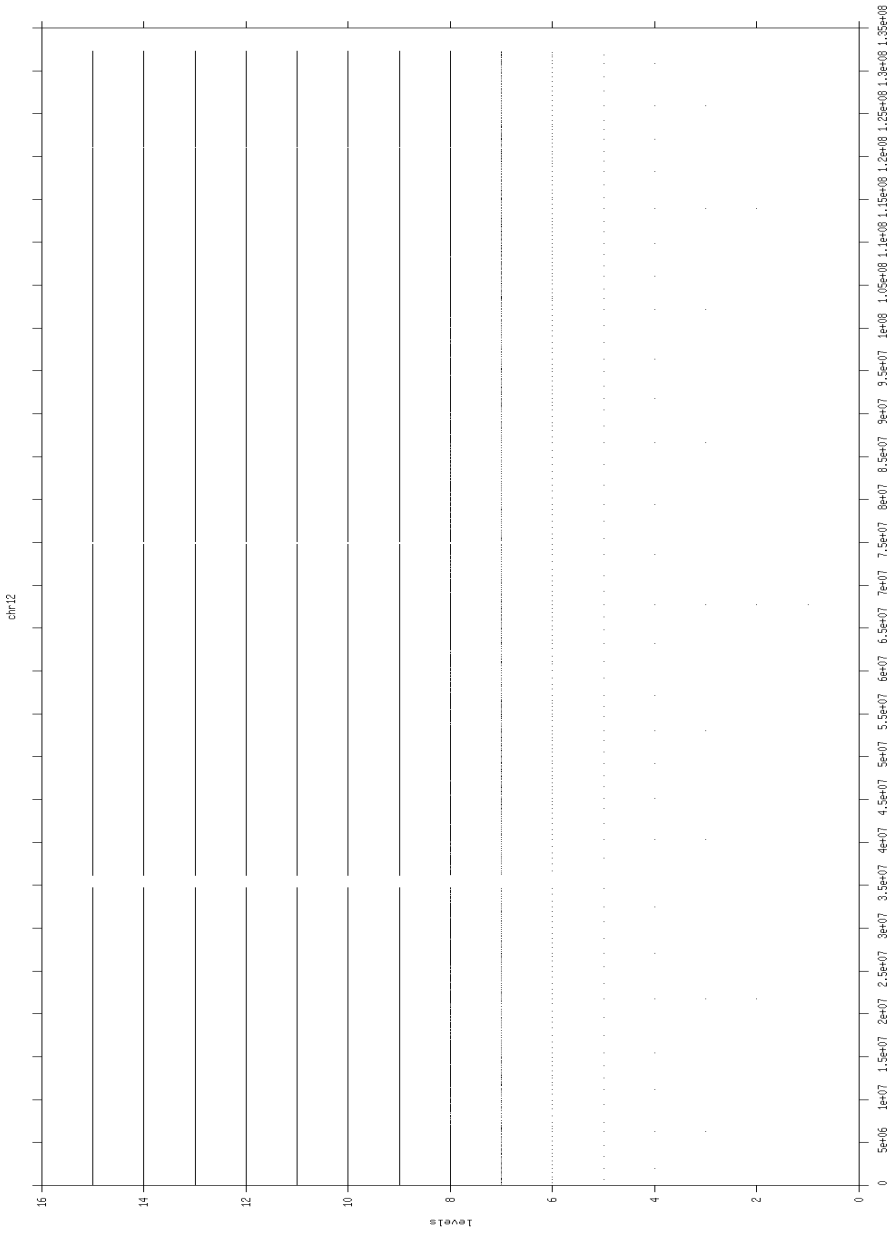


Figure 16: graphical representation of the partitions on chromosome 12

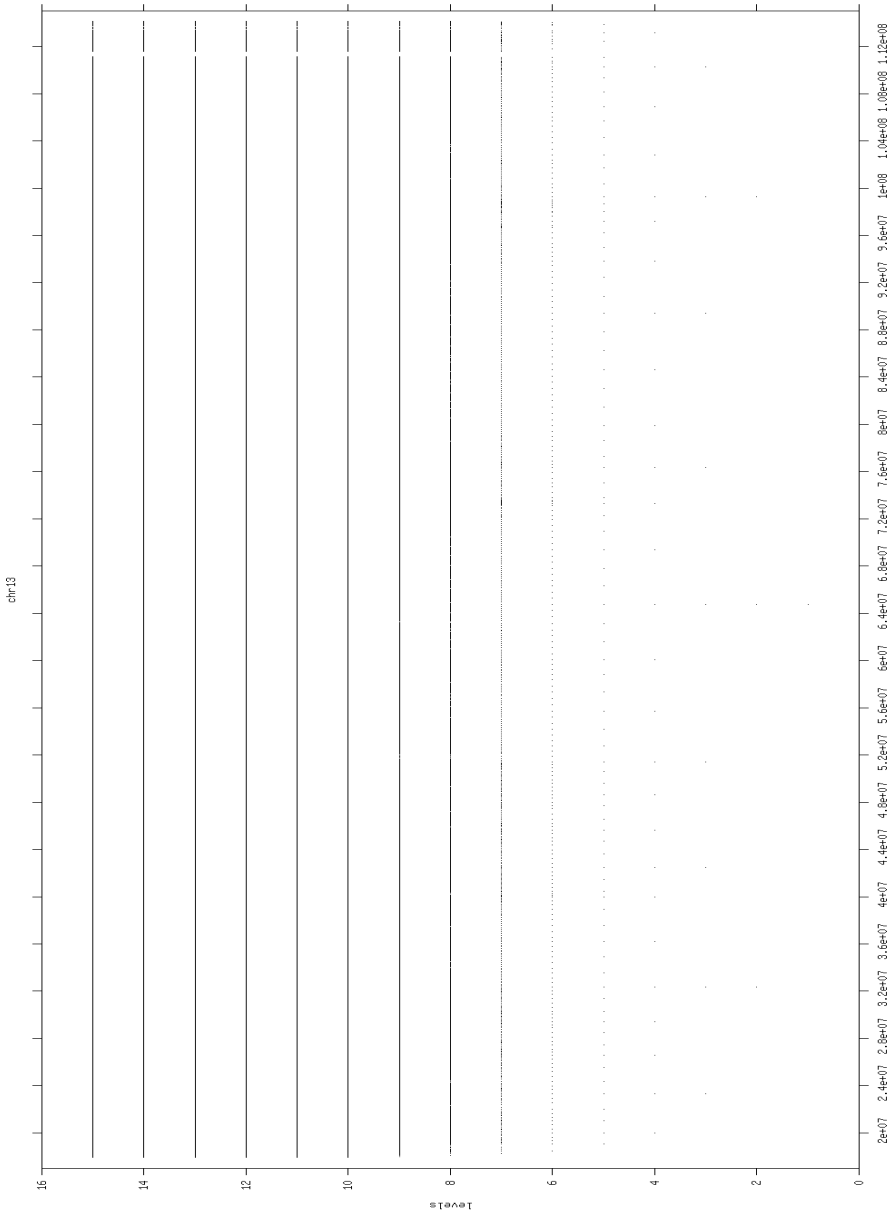


Figure 17: graphical representation of the partitions on chromosome 13

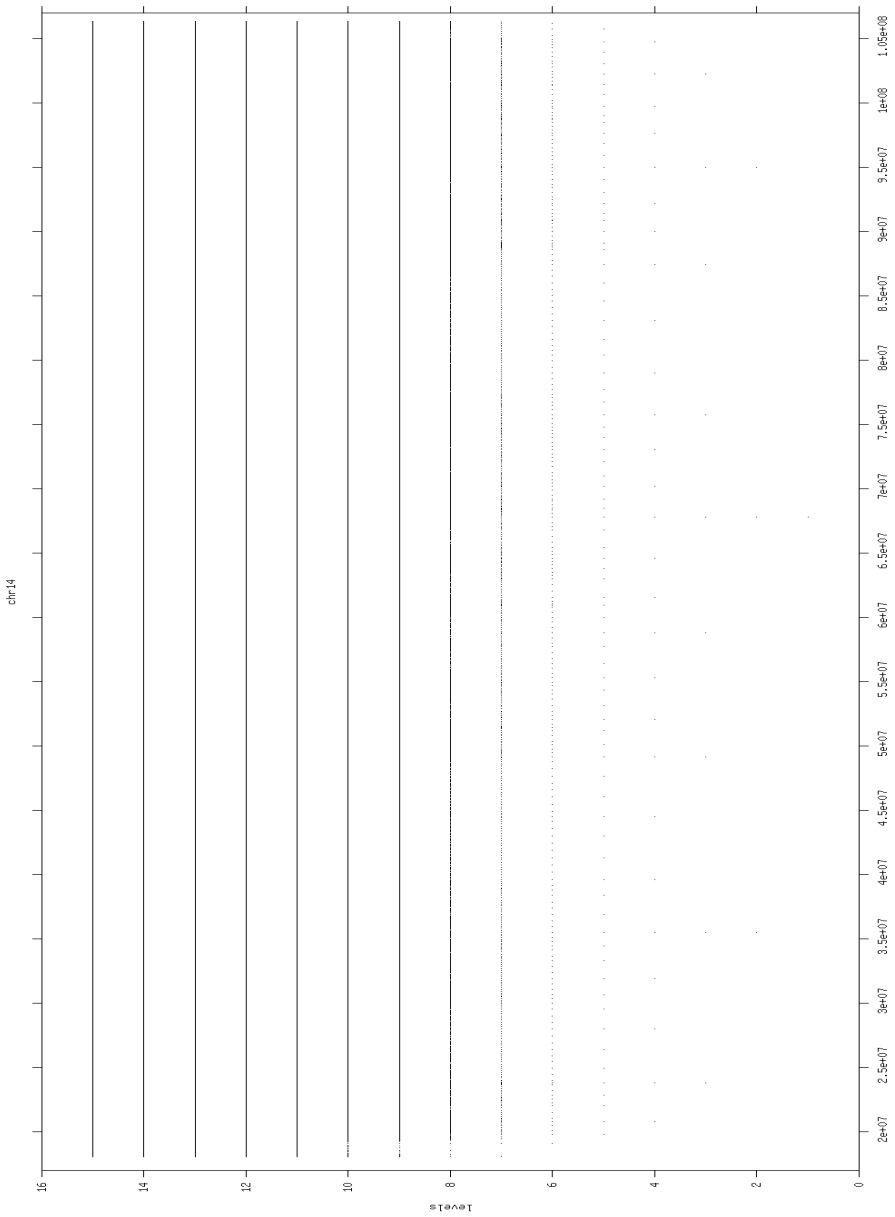


Figure 18: graphical representation of the partitions on chromosome 14

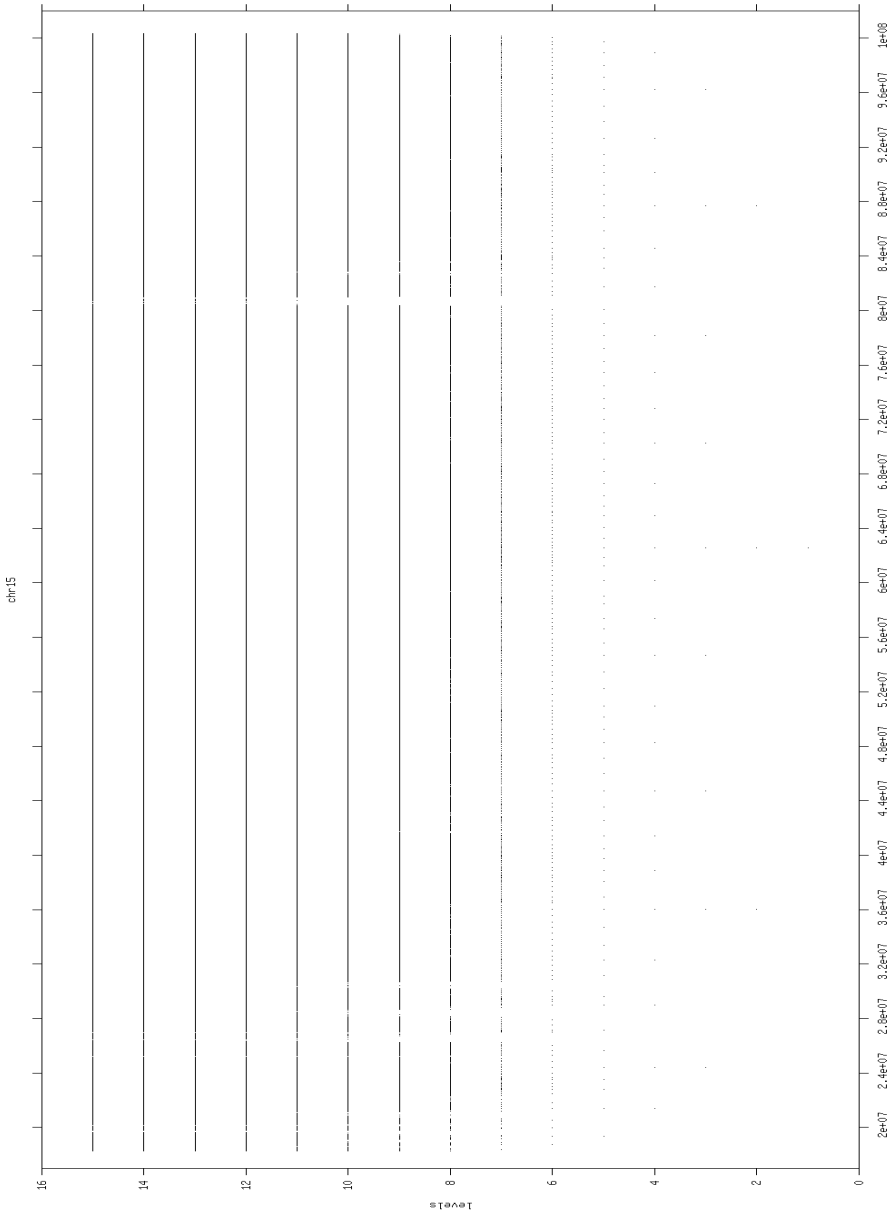


Figure 19: graphical representation of the partitions on chromosome 15

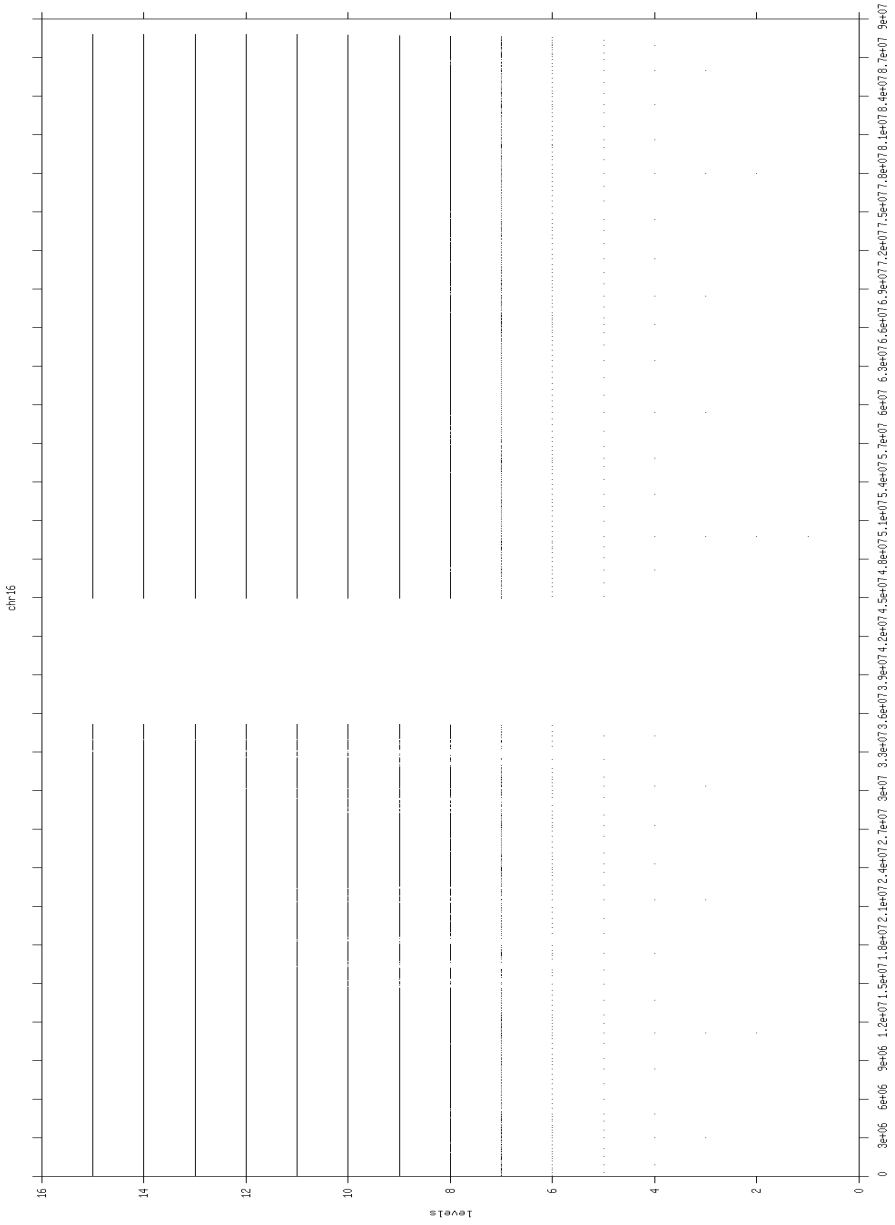


Figure 20: graphical representation of the partitions on chromosome 16

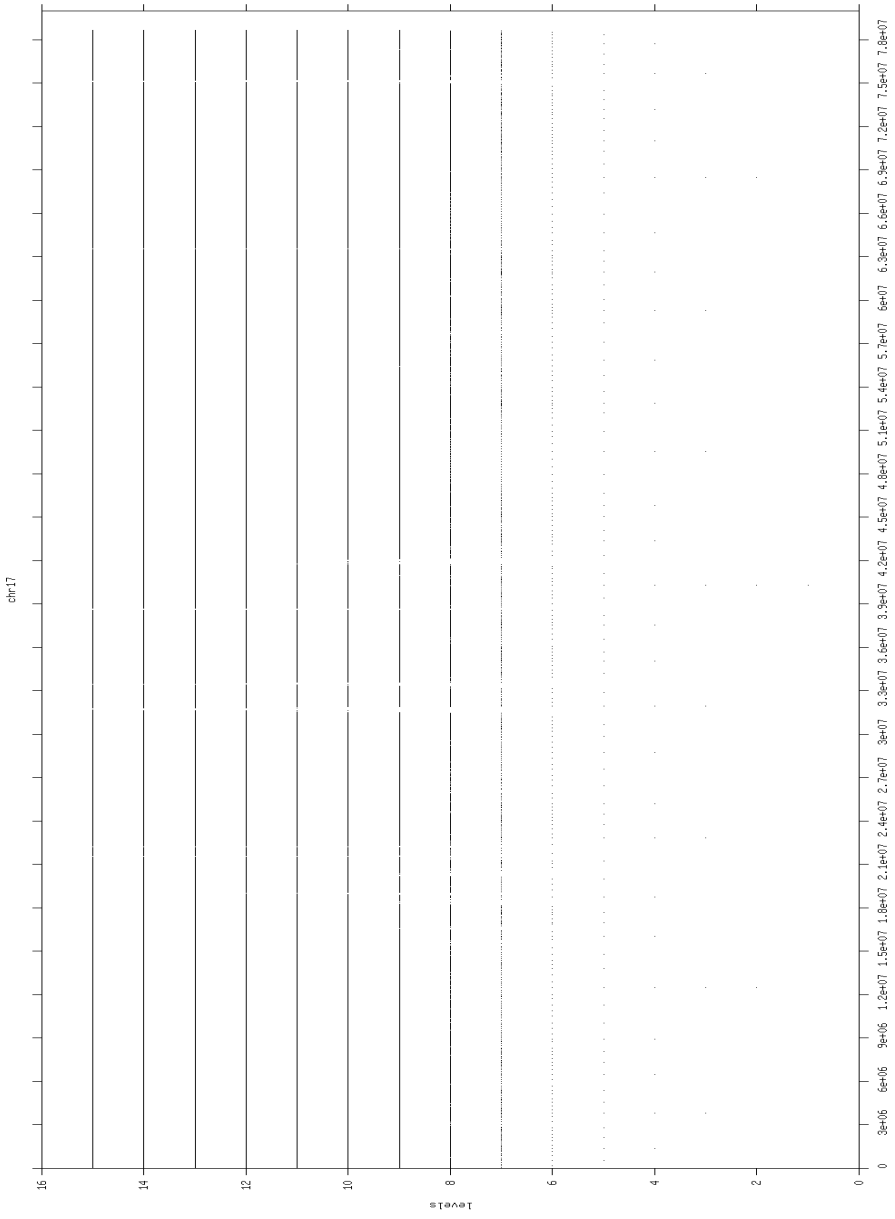


Figure 21: graphical representation of the partitions on chromosome 17

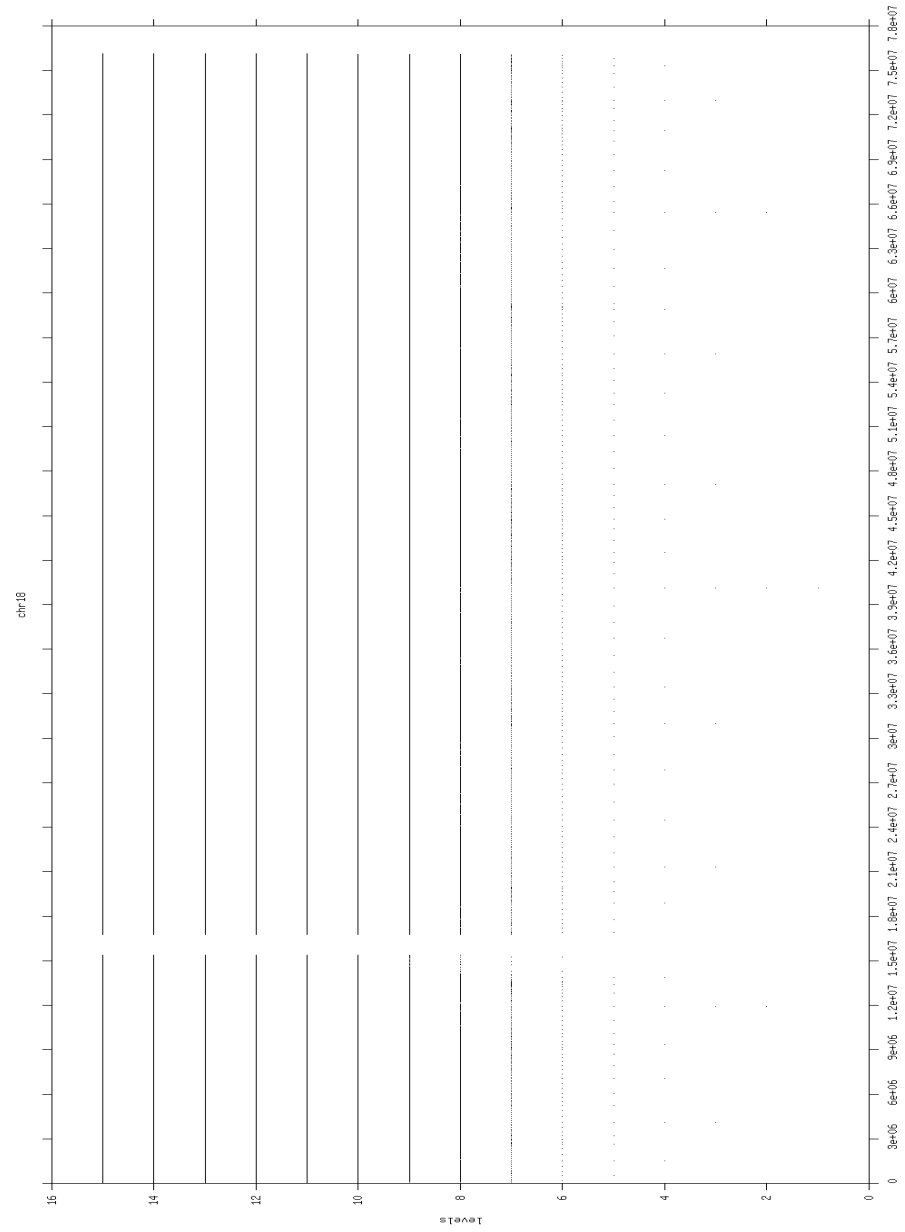


Figure 22: graphical representation of the partitions on chromosome 18

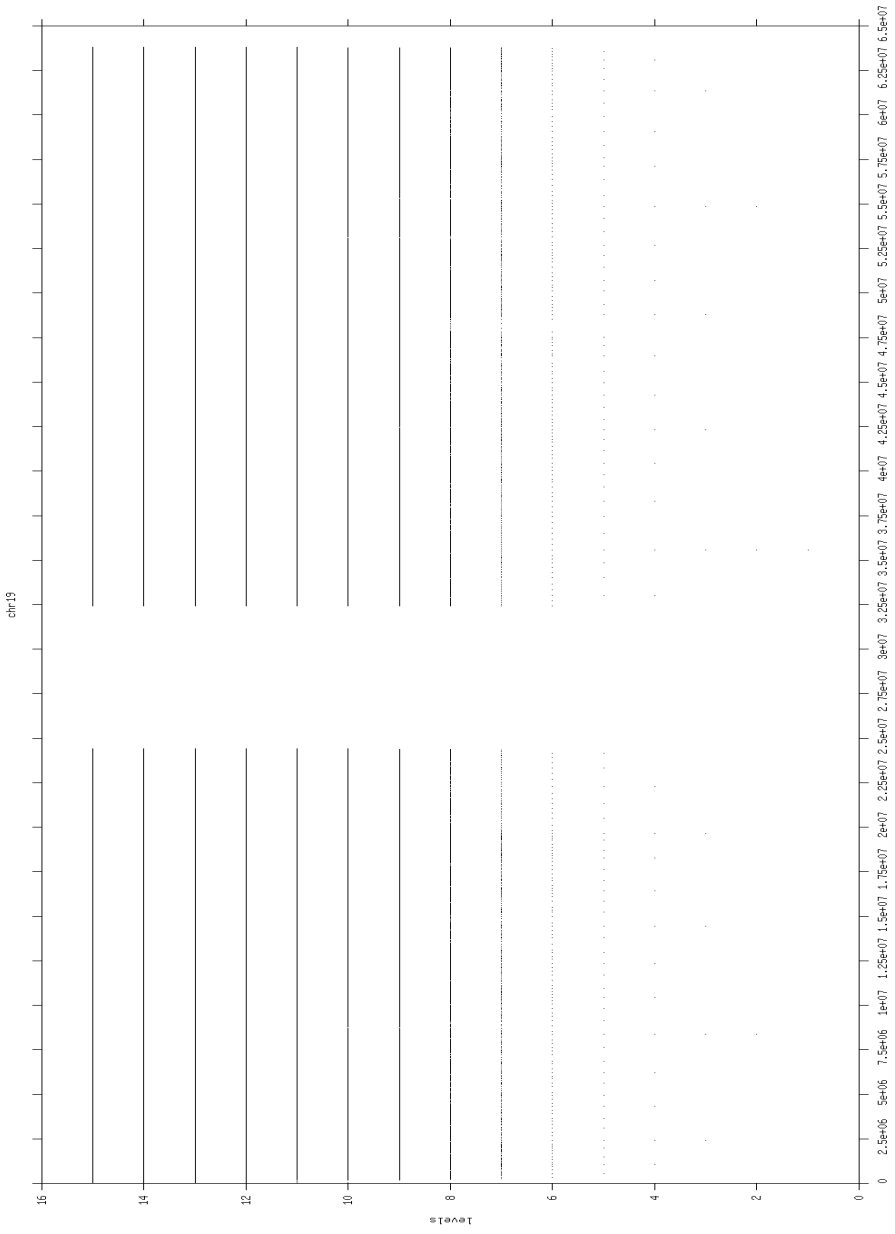


Figure 23: graphical representation of the partitions on chromosome 19

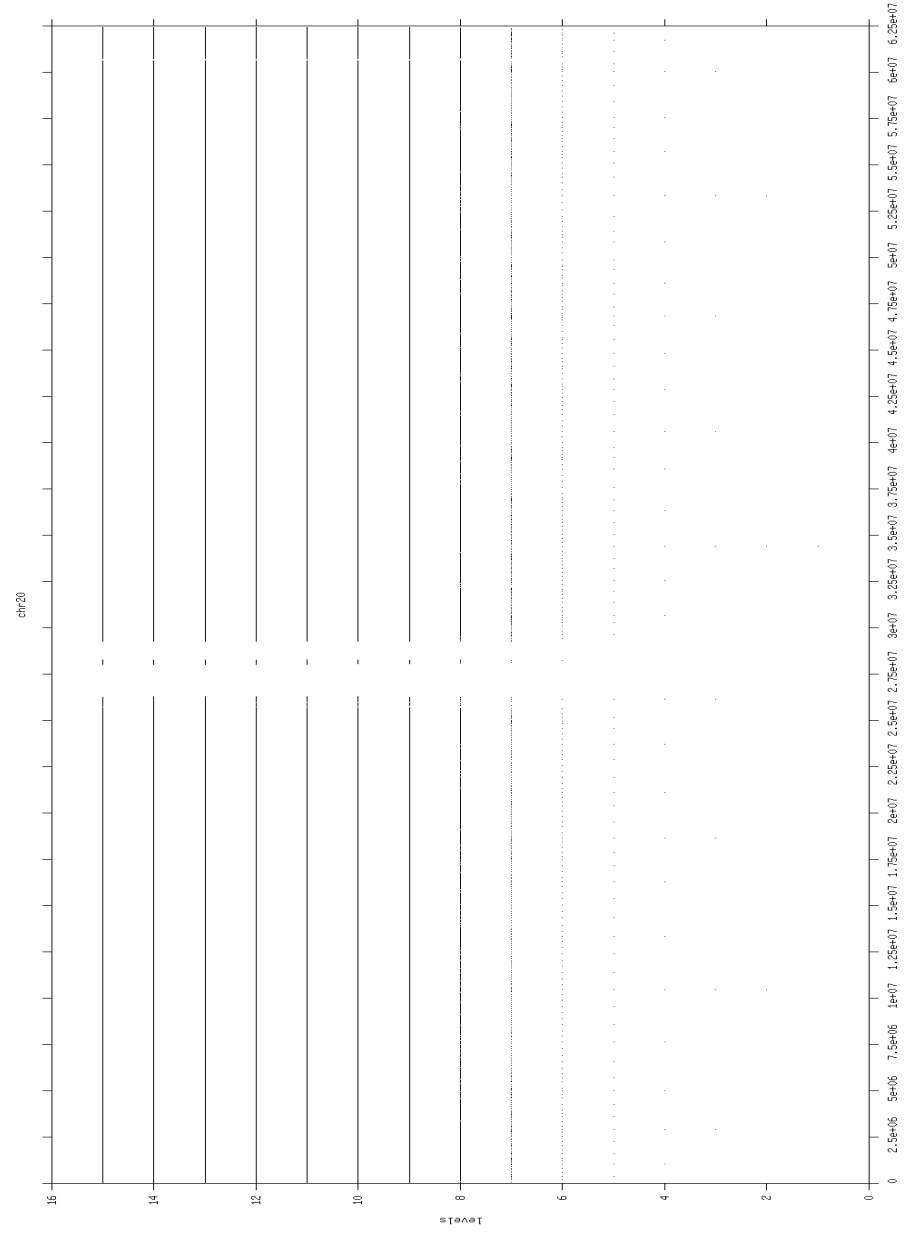


Figure 24: graphical representation of the partitions on chromosome 20

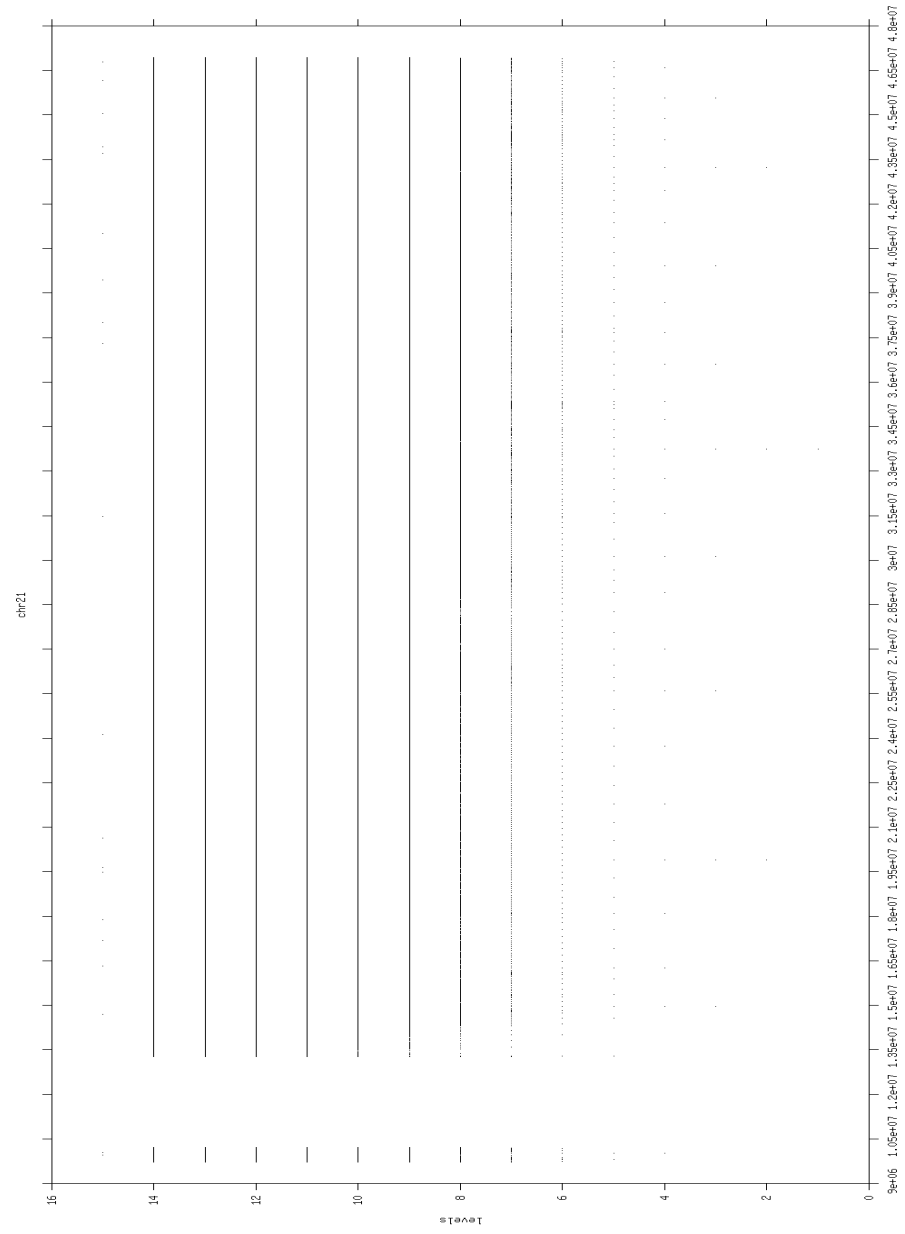


Figure 25: graphical representation of the partitions on chromosome 21

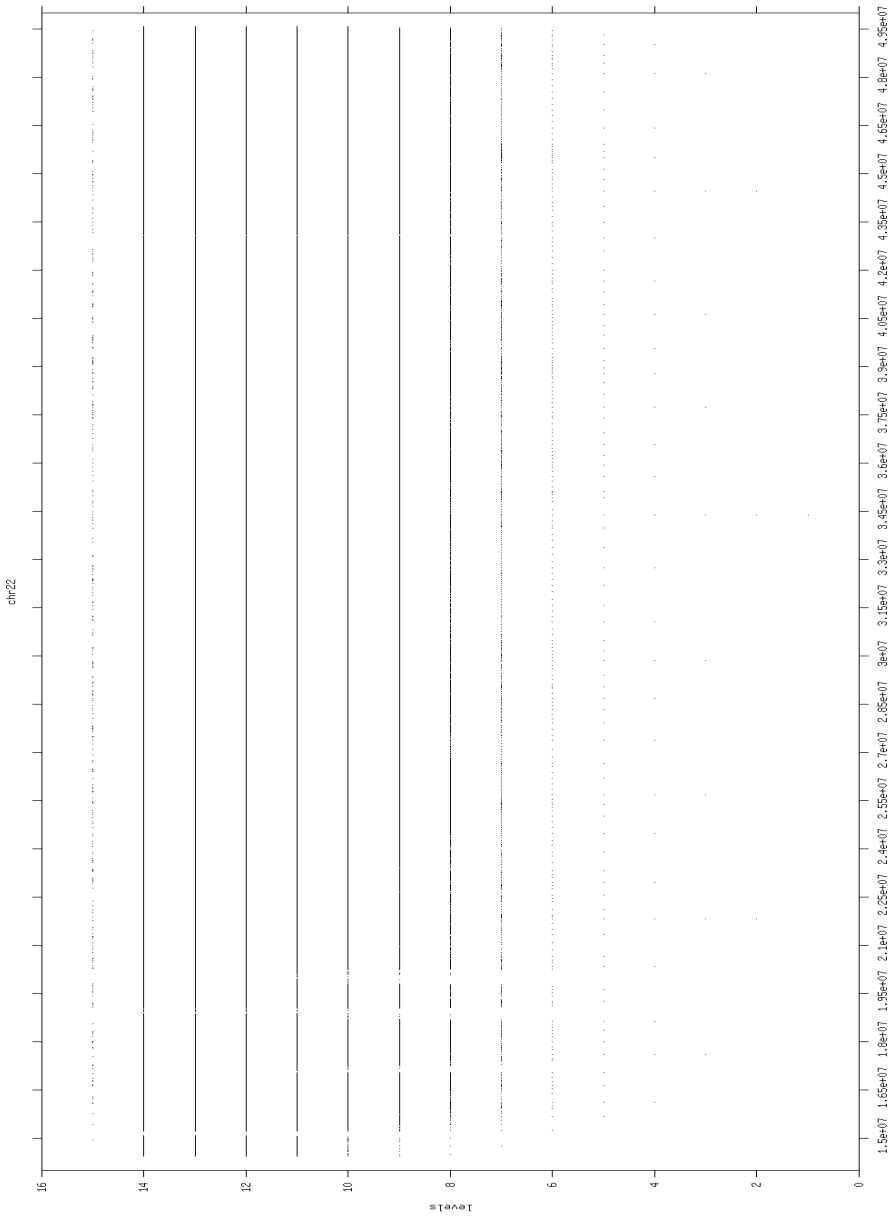


Figure 26: graphical representation of the partitions on chromosome 22

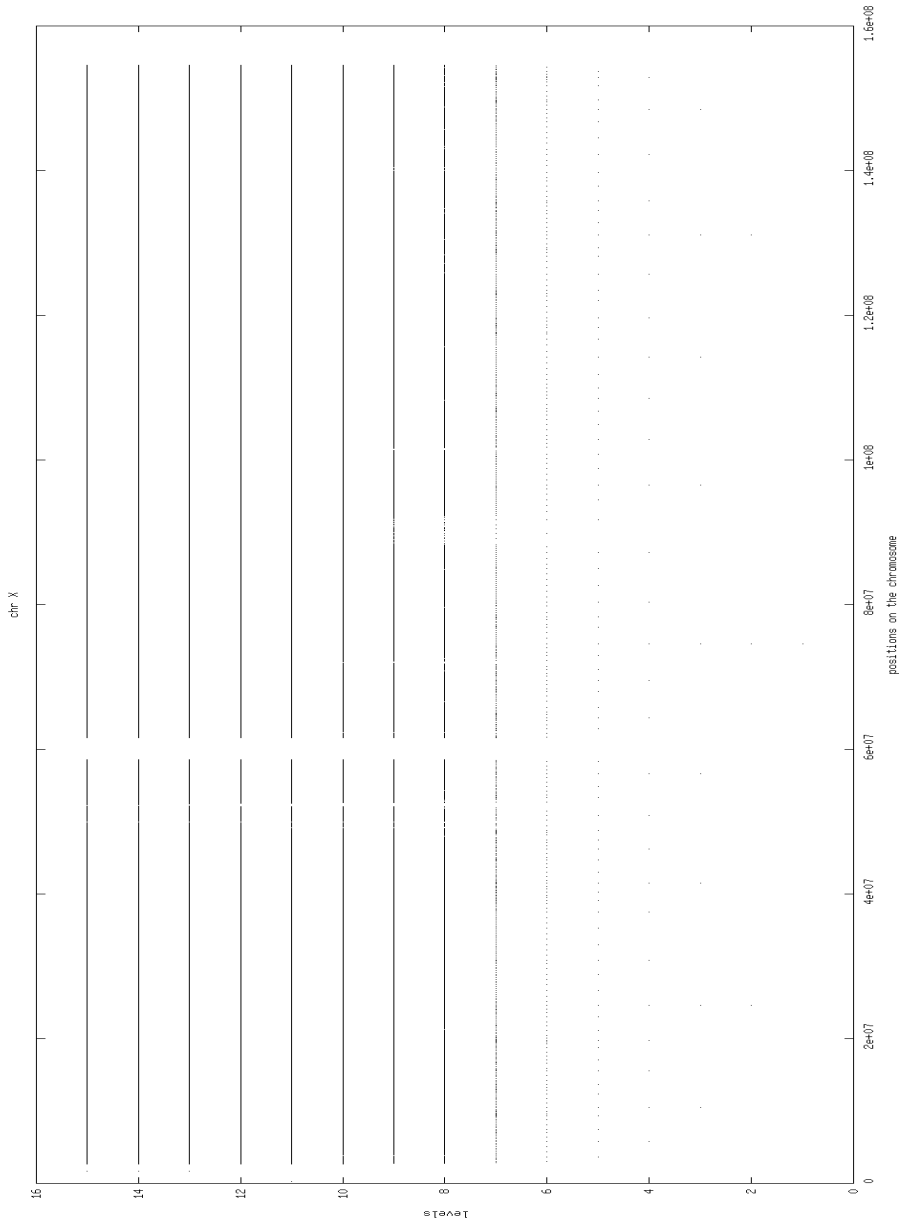


Figure 27: graphical representation of the partitions on chromosome X

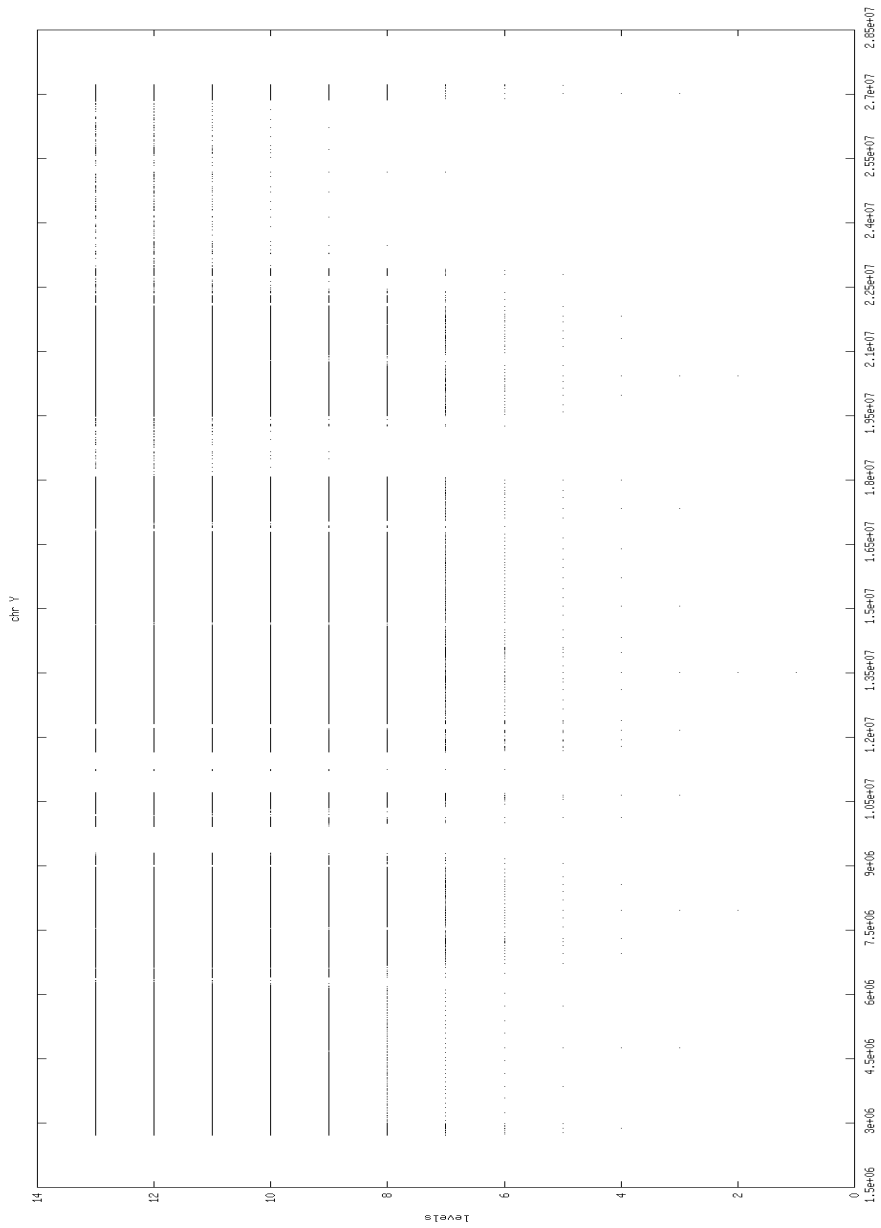


Figure 28: graphical representation of the partitions on chromosome Y