The Genome Project: Looking for Similarities in Human Chromosomes using Elongation Matrices

Stefano Angeleri, Luca Battisti, Alessandro Menti, Mattia Zago

Abstract

Infogenomics is a relatively new branch of bioinformatics that studies genomes from a quantitative point of view. Genomes are viewed as strings of a formal language $\Gamma^* = (\{A, C, G, T\})^*$ whose alphabet consists of the four DNA bases (adenine, cytosine, guanine and thymine) and on which statistical and informational analyses are made, so as to extract possible substructures, words and formal grammars generating the language. Assuming that well-chosen informational indexes correspond to defined biological properties, this approach could help scientists to understand the links between the genotype of an organism and any of its phenotipic expression, as well as comparing the expression of a same genomic word or set of words in different species.

Our project aimed to extract possible dictionaries from the human genome by computing significative informational matrices (such as string elongation and multiplicity elongation matrices) using a custom-built distributed analysis software based on Hadoop, plotting the results and extracting significative patterns emerging from the data.

1 Informationally significant matrices

We consider the following matrices as significant indicators to be used for our analysis.

Definition 1 (String elongation matrix)**.** Let Γ be an alphabet (whose elements are considered in the order they are enumerated in that set) and α be a string on Γ^*. The *string elongation matrix (SEM)* for α is a matrix where each coefficient $\alpha_{i,j}$ is the character immediately following the j-th occurrence in α of the i-th character in Γ (if such a character exists).

Definition 2 (Multiplicity elongation matrix)**.** Let Γ be an alphabet (whose elements are considered in the order they are enumerated in that set) and α be a string on Γ^*. The *multiplicity elongation matrix (MEM)* for α is a matrix whose coefficients $\alpha_{i,j}$ are the number of occurrences of the j-th character of Γ that follow the occurrences of the i-th character of Γ immediately.

Definition 3 (Boolean elongation matrix)**.** Let Γ be an alphabet (whose elements are considered in the order they are enumerated in that set) and α be a string on Γ^*. The *boolean elongation matrix (BEM)* for α is a matrix whose coefficients $\alpha_{i,j}$ are:

- 1 if, in the string α, at least an occurrence of the i-th character of Γ is immediately followed by an occurrence of the j-th character of Γ;
- 0 otherwise.

Remark 1. Given a string elongation matrix for a string α, the multiplicity elongation matrix can be obtained by counting the occurrences of each character of Γ appearing in each row and replacing each row of the SEM with the counts obtained this way; given a multiplicity elongation matrix for a string α, the boolean elongation matrix can be obtained by replacing every non-zero coefficient with 1. Both tasks are linear in the number of coefficients (or, equivalently, quadratic in the cardinality of Γ).

Definition 4 (Dictionary elongation matrix)**.** Let Γ be an alphabet (whose elements are considered in the order they are enumerated in that set) and α be a string on Γ^*. The *dictionary elongation matrix (DEM)* for α is a matrix whose rows are the rows of the string elongation matrix for α in which each character appears at most once, no empty coefficients precede a character and, for each pair of non-empty coefficients α_{i,j_1}, α_{i,j_2}, $j_1 < j_2$ if (and only if) the first occurrence of α_{i,j_1} in the i-th row appears before the first occurrence of α_{i,j_2} in the i-th row.

Example 1. If $\Gamma = \{A, B, C, D, E\}$ and $\alpha = ABEABDEBACEADDDEABEE$, the SEM for α is:

$$\begin{bmatrix} B & B & C & D & B \\ E & D & A & E & \\ E & & & & \\ E & D & D & E & \\ A & B & A & A & E \end{bmatrix}$$

The MEM for α is:

$$\begin{bmatrix} 0 & 3 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 & 2 \\ 3 & 1 & 0 & 0 & 1 \end{bmatrix}$$

The BEM for α is obtained by replacing the non-zero coefficients with 1:

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

The DEM for α is:

$$\begin{bmatrix} B & C & D \\ E & D & A \\ E & & \\ E & D & \\ A & B & E \end{bmatrix}$$

The key to our approach is noticing that the multiplicity elongation matrix can give us informations about the most frequent sequences of hexamers that appear in the genome by:

- working with an alphabet Γ' containing every possible hexamer or, more formally, $\Gamma' = \{A, C, G, T\}^6$, where, as usual,

$$\Gamma^n = \begin{cases} \{\varepsilon\} & \text{if } n = 0, \\ \{ax\colon a \in \Gamma, x \in \Gamma^{n-1}\} & \text{otherwise;} \end{cases}$$

- approximating MEMs of sequences of hexamers of arbitrary length with a succession of MEMs relating each hexamer in the sequence to the next one;
- assuming that greater MEM coefficients and, thus, higher numbers of times a hexamer appears in the genome, correspond to an increased probability that it constitutes a meaningful word in the genomic dictionary.

Similar considerations can be made for boolean elongation matrices.

In order to take into account any possible mutations or other factors which may displace the initial symbol of the hexamer, we introduce a sliding window (that is, an initial offset from which we started processing the data) varying from zero (the scenario where we optimistically assumed that no mutations were present) to five (the maximum possible offset, as greater ones can be considered as an initial insertion of spurious hexamers plus an additional offset of no more than five bases). We then apply this sliding window to the genome we analyze six times (each one for every possible value of the parameter), extract the informationally relevant matrices and combine them by summing their coefficients (we call this operation a *matrix union*).

2 Algorithm and first version of the software architecture

We employed the Human Genome dataset provided by the Genome Reference Consortium [3], in the commonly used textual FASTA format, as the reference data source for our analysis. The first version of the custom software we designed consisted of the following modules:

1. a FASTA preparser and rebuilder to remove spurious characters from input data and to rebuild a FASTA formatted file from a SEM;
2. a Hadoop mapper extracting hexamers from a portion of the preprocessed file and creating partial SEMs;
3. a Hadoop reducer that combines the partial results obtained from the mapper, generating the complete SEMs;
4. a postparser that derives the other informationally significant matrices from the SEMs.

We ran the software six times per chromosome, deleting a single DNA base every time (applying the sliding window). The resulting data was then plotted and analyzed using custom-made spreadsheets and scripts.

2.1 The FASTA preparser

The FASTA file format specification mandates that each file must begin with a single line description of the encoded sequence and allows it to include, beside the expected nucleotide sequences in the usual IUB/IUPAC format, hyphens and dashes (to represent gaps of indeterminate length) as well as numerical digits; also, lowercase and uppercase letters may be used inconsistently [4]. For this reason, raw data is processed by a custom preparser/sanitizer that;

- removes the initial description;
- removes all characters except the ones corresponding to nucleotides, also converting them to uppercase;

- splits the data in subblocks suitable for parallel processing, letting the user choose their dimension (we did not exceed 100000 hexamers per block to balance the need to avoid initializing too many matrices, depleting memory, with the one to keep the chunk size relatively low to get feedback as data is elaborated).

2.2 The mapper and the reducer

We use the MapReduce model [2] and the Hadoop framework [1] to parallelize the matrix extraction from the sanitized file.

The idea at the base of the MapReduce model is to apply a *divide-and-conquer* strategy to data organized in $(key_1, value_1)$ pairs. The input file is split in subblocks by a master node in a cluster; each of them is sent to a slave node. Every slave node runs a `map` function on its block (computes the partial result of the parallelized algorithm) and returns a list of ordered pairs $(key_2, value_2)$ consisting of new keys and of the associated values (results); the master node then collects these lists, grouping them by key_2, and sends each group (along with all associated values) to a slave node. The slave node runs the `reduce` function on the assigned key_2 and passes the result back to the master node; the latter aggregates all the partial results and gives back the final output.

We applied this model only to the SEM generation procedure since this is the most time consuming task (by Remark 1 and by noting that we can generated a DEM from a SEM by removing duplicates from each row, preserving the order of the coefficients, we can obtain MEMs and SEMs using low-complexity algorithms). Our program works as follows.

1. Hadoop gives each mapper a substring of the sanitized FASTA data (the amount of subblocks is configurable at runtime).

2. The mapper initializes a matrix with 4096 rows, one for each possible hexamer; each row is an empty list. After that, the mapper extracts all hexamers and their immediate successors from the raw data and stores all the immediate successors in the appropriate list; in the end, it outputs a list of pairs (*hexamer*, *encoded list of immediate successors*). More formally:

```
MAPPER(key, value, c)
 1  matrix = empty SEM_MATRIX
 2  while value.length > 5
 3      first = CREATE-HEX(value[0..5])
 4      value = value[6..]
        // If we can't extract a second hexa-
        // mer, keep track of the residue.
 5      if value.length < 6
 6          matrix.lastUnmatched = first
 7          matrix.residue = value
 8          break
 9      next = CREATE-HEX(value[0..5])
10      PUSH(matrix, first, next)
11  for i = 0 to matrix.rows
12      row = GET-ROW(i)
13      if row is not empty
14          hex = GET-HEXAMER-TEXT(i)
15          rowText = GET-ROW-TEXT(i)
16          CONTEXT-WRITE(c, hex, rowText)
```

3. The reducer gets a hexamer and a set of encoded lists of immediate successors as input; it sorts the set (keeping the order of the successors into account), decodes the lists and generates the row of the string elongation matrix corresponding to the given hexamer. More formally:

```
REDUCER(key, values, c)
 1  e = CREATE-HEX(key)
 2  rowName = "FINAL-"
        + HEXAMER-GET-POSITION(e) + "-"
        + key;
 3  result = CREATE-SEMROW(rowName)
    // Sort the set.
 4  list = new empty list
 5  it = values.iterator
 6  n = it.head
 7  while n ≠ NIL
 8      LIST-ADD(list, CREATE-TEXT(n))
 9      n = n.tail
10  LIST-SORT-BY-SEMID(list)
11  for i = 0 to list.size
12      row = list[i]
13      PUSH(result, row)
14  rowText = GET-ROW-TEXT(result)
15  CONTEXT-WRITE(c, rowText)
```

2.3 The FASTA postparser

The postparser is split in two parts:

MergeReducer: this procedure, which can be omitted optionally for testing purposes, concatenates the intermediate files generated by the instances of the reducer to generate the final merged file;

decodeResult: takes as input the final merged file, calculates the rows of the other informationally relevant matrices (MEM, BEM, DEM) for each hexamer and stores them in separate files, then merges them by concatenating the rows to obtain the final matrices.

3 Improving performance

We were forced to execute the first version of our software in a virtual machine environment on our own systems since the departmental Hadoop cluster, during our tests, suffered from heavy slowdowns and killed the mapper processes after 13 hours of computation, whereas the same code, when run on our hardware, computed the results in just a few hours (or less) per chromosome (we suspect the issue to be caused by low resources, specifically processing power and memory, as qualitative tests showed that better CPUs and higher amounts of available RAM are proportional to the execution time).

To drastically reduce the footprint, we implemented a design change, namely passing triplets of the form (*hexamer*, *offset*, *next hexamer*) between the mapper and the reducer (instead of whole matrix rows) where:

- *hexamer* is the current hexamer;
- *offset* is the offset of the hexamer inside the file;
- *next hexamer* is the hexamer immediately following the current one.

3.1 Reading records and performing the mapping phase

Data is fed to the new mapper using an ad-hoc file input format [5, p. 178], named `FASTAFileInputFormat`, and a custom record reader [5, p. 179], named `FASTAFileRecordReader`, that performs the function of the FASTA preparser (removing the description line and spurious characters from the input files and converting all letters to uppercase). The record reader takes as input a *file split*[1] and outputs (*key*, *value*) pairs where:

- the key is the start position of the emitted chunk in the input file;
- the value is a genome chunk not exceeding the maximum number of hexamers per chunk specified in the driver code with the `--hex` command line parameter;
- the outputted genome chunk respects the file split boundaries assigned by the Hadoop framework (as an exception, if a split ends at a file position that is not a multiple of six, the reader completes the hexamer before returning the chunk).

More formally, the `nextKeyValue` method of the record reader (called by Hadoop to get the next available pair for processing) uses the following algorithm:

```
NEXTKEYVALUE(hexPerBlock)
 1  firstValidCharSeen = FALSE
    // We use a StringBuffer sb to speed up string
    // concatenation.
 2  sb = ""
 3  remaining = hexPerBlock
 4  while F.pos < F.end and remaining > 0
 5      if remaining == hexPerBlock
 6          if outputHex == NIL
 7              outputHex = ""
 8          else
 9              SBAPPEND(sb, outputHex)
10              remaining = remaining − 1
11      for i = 0 to 5
12          repeat
13              c = NEXTCHARTOUPPERCASE()
14              if c == EOF
15                  break
16              if c is a valid base
17                  outputHex[i] = c
18                  break
19          until c <> EOF
20          if firstValidCharSeen == FALSE
21              key = F.pos
22              firstValidCharSeen = TRUE
23      SBAPPEND(sb, outputHex)
24      remaining = remaining − 1
25  if sb.length == 0
26      value = NIL
27      return FALSE
28  else
29      value = SBTOSTRING(sb)
30      return TRUE
```

The mapper just reads the genome chunks and emits (*hexamer*, *offset*, *next hexamer*) triplets:

[1] A *file split* is a part of the original input file. File splits are automatically created for supported input formats by the Hadoop framework so that relevant portions of one or more input files can be associated with map task slots taking data locality into account.

```
MAPPER(key, value, c)
 1  if key.length < 12
 2      break
 3  currentPosition = 0
 4  firstHex = value[0..5]
 5  nextHex = value[6..11]
 6  while TRUE
 7      CONTEXT-WRITE(c, CREATE-HEX-OFFSET(
            CREATE-HEX(firstHex), key),
            CREATE-HEX(nextHex))
 8      currentPosition = currentPosition + 6
 9      if currentPosition + 12 >= key.length
10          break
11      firstHex = nextHex
12      nextHex = value[currentPosition + 6..
            currentPosition + 11]
```

3.2 Partitioning/grouping triplets and emitting SEM rows

To generate the SEM rows we now need to sort and group the data so that triplets with the same hexamer in the key are sent to the same reducer and are sorted by offset. We use the *secondary sorting* design pattern [8, p. 276] for that purpose.

Given a set of records consisting of a composite key and a value, secondary sorting groups the records by a part of the key (chosen by the developer) and sorts each group before sending it to the appropriate reducer. This design makes use of three components:

1. a *composite key comparator*, or *sort comparator*, used to perform the sorting. The class should extend `RawComparator` and should perform a sequence of comparisons such that the single parts making the key are evaluated from the most generic to the most specific one;

2. a *natural key grouping comparator*, or just *grouping comparator*, used to group records according to the natural (or most generic) key;

3. a custom *partitioner* that partitions the records according to their natural key, so as to ensure that records having the same natural key are sent to the same reducer.

In our case:

- the composite key is the (*hexamer*, *offset*) pair created by the mapper;

- the composite key comparator will be the default one used by the Hadoop framework (since it compares keys using their `compareTo` method and we have overridden it in `HexamerOffsetPair` to compare pairs first by hexamer and then by offset, there is no need to implement a custom class);

- the natural key grouping comparator will be a custom class (`HexEmitGroupingComparator`) comparing just the "hexamer" part of the key;

- the partitioner will be, again, a custom class (`HexEmitPartitioner`) that gets the hash code of the hexamer in the key and chooses the correct partitioner according to it.

All these properties are set via appropriate methods (`setPartitionerClass`, `setGroupingComparatorClass`) in the driver code.

This way, all records with the same hexamer in the key are sent to the same reducer and sorted by offset, so the only thing we need to do in the "reduce" stage is to concatenate all values and output the SEM row:

```
REDUCER(key, values, c)
1  result = CREATE-SEMROW()
2  for i = 0 to list.size
3      PUSH(result, list[i])
4  CONTEXT-WRITE(c, GET-HEXAMER(key), result)
```

The only operation left at this point is sorting the emitted SEM rows lexicographically to generate the final string elongation matrix (and the other derived matrices).

3.3 Sorting SEM rows with a second Hadoop job

Hadoop itself provides no guarantees with regards to the order of the reducer output. This conflicts with the definition of (approximated) SEM, where rows are ordered according to the natural, lexicographic hexamer order. To sort the output, we chained a second Hadoop job with the following properties:

- the mapper and the reducer are not specified, so that the framework will use the built-in "trivial" ones (by default, MapReduce sorts input records by their keys before passing them to the reducer);

- the number of reduce tasks is set to one since the final matrix creation is a sequential operation;[2]

- we make use of multiple output formats [8, p. 251] to generate all the informational matrices at the same time.

[2] We deemed parallelizing this job a non-trivial task with a limited potential performance gain.

3.4 Execution statistics

As a testbed, we ran the two versions of our software in different environments to evaluate their performance (with regard to execution time and RAM usage). Specifically, the three test configuration we adopted are:

1. first (unoptimized) version in a virtual machine environment (Oracle VirtualBox VM running Ubuntu 12.04.3 X86_64 with the 3.2.0-53-generic kernel, with 6000 MB of available RAM and 4 CPUs);
2. first version on the departmental cluster;
3. second (optimized) version on the departmental cluster.

The results can be seen in Figure 8. Some key points are:

- The first, unoptimized version of our code has an irregular RAM usage ranging between 55 and 80 percent, with many spikes, while the second one has a more regular trend (although the amount of used memory is much higher, close to 100%).
- For the old code, execution times are significantly higher on the cluster (107 seconds) than in the VM environment (10 seconds), probably due to swapping phenomena; the data for the old software running in our VM and the new one running on the cluster, on the contrary, are similar, save for a minor difference after the 270000 hexamer mark.

	Most frequent		Less frequent	
Chr1	129540	AAAAAA	139	TACGCG
Chr2	131124	AAAAAA	132	TACGCG
Chr3	108039	AAAAAA	98	CGCGTA
Chr4	102654	AAAAAA	87	TACGCG
Chr5	97003	AAAAAA	85	CGATCG
Chr6	93612	AAAAAA	91	CGCGTA
Chr7	90262	AAAAAA	92	TACGCG
Chr8	76653	AAAAAA	81	CGCGTA
Chr9	68521	AAAAAA	63	TACGCG
Chr10	73873	AAAAAA	83	TACGCG
Chr11	69461	AAAAAA	83	TACGCG
Chr12	77403	AAAAAA	68	CGTACG
Chr13	52024	AAAAAA	47	TACGCG
Chr14	50415	AAAAAA	44	CGCGTA
Chr15	48582	AAAAAA	52	TACGCG
Chr16	49091	AAAAAA	70	TCGACG
Chr17	39765	AAAAAA	39	TACGCG
Chr18	41792	AAAAAA	73	CGTACG
Chr19	41792	AAAAAA	73	CGTACG
Chr20	31488	AAAAAA	34	CGTACG
Chr21	18980	AAAAAA	21	TACGCG
Chr22	21043	AAAAAA	35	CGTCGA
ChrX	81032	AAAAAA	72	CGCGTA
ChrY	13680	AAAAAA	6	TACGCG
Rand01	27670	AAATAA	2302	CGCGCG
Rand19	6962	TAAAAA	542	CGCCCC

Table 1. List of notable (most and less recurring) hexamers derived from the string elongation matrix.

4 Processing and results

We ran our software on the Y chromosome as a testbed to check, refine and expand our analysis and also generated a graphical representation of its MEM. We then executed our program on the whole human genome, adding some indicators and distributions we deemed significative.

We extracted the maximum, minimum and average values for each row of each SEM, as well as for the SEM of each chromosome. We generated a table to keep track of the number of occurrences of each possible hexamer in every chromosome, as well as the number of hexamers per chromosome, then we normalized the data so as to look at the *relative* number of occurrences.

The SEM normalized count plot we obtained (Figure 1) shows that the number of hexamers per chromosome is approximately constant (about 0,34 × count, where count is the number of hexamers per chromosome); there are two notable exceptions to this behaviour, namely chromosomes 17 and 19, for which the number is substantially higher (about 0,41 × count and 0,46 × count, respectively). An elaboration of the *min value* indicator (defined as the minimum number of times any hexamer appears in the chromosome) showed a similar pattern, with chromosomes 16, 19, 22 and Y being notable outliers. We also counted the most and less frequent hexamers for every chromosome, as well as for two random sequences with the same length as chromosomes 1 and 19 (named, respectively, Rand01 and Rand19), and found an exceptionally high number of occurrences of the hexamer AAAAAA in all non-random chromosomes. Five hexamers make the "less recurring" list (CGTACG, TACGTG, CGCGTA, CGATCG and TCGACG). The details are in Table 1.

We then analyzed the multiplicity elongation matrix, limiting the analysis to the first chromosome (a test run on some other chromosomes was not judged very useful, as the results were qualitatively the same). A plot of the MEM for two hexamers, AAAAAA and TTTTTT, reveals that the values in the matrix tend to accumulate in an interval between 0 and 100, with noticeable peaks up to approximately 200/300 and

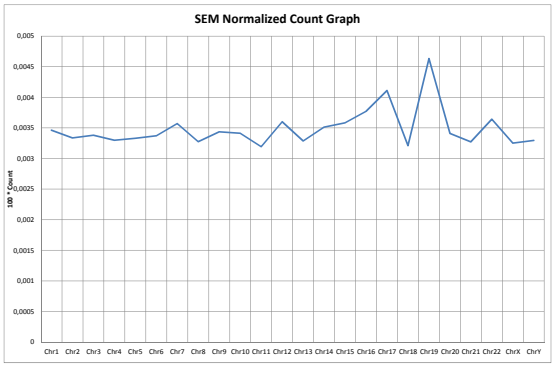


Figure 1. A plot of the SEM normalized count.

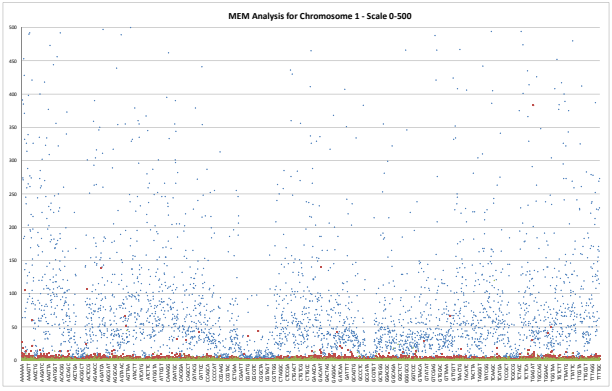


Figure 2. A plot of the multiplicity elongation matrix for chromosome 1 in the [0,500] range.

some outliers reaching 500 (see Figure 2). The same plot, using a logarithmic scale, shows a much more linear trend. Some sequences (ACGAGA, CCGATC, CGAGAA-CGTGCA, GCGCAG, TCGCCT) have low values in the plot, meaning that the hexamers in the series do not follow them.

We drew most conclusions and derived a number of indicators from the boolean elongation matrix.

Definition 5 (Hexamer coverage distribution)**.** Let c be a chromosome, h a hexamer and BEMc the boolean elongation matrix for c. The *hexamer coverage distribution* relative to c and h is defined as follows:

$$\mathrm{HC}(c,h) = \frac{1}{4096}\sum_{i=1}^{4096} \mathrm{BEM}c_{c,h}$$

Definition 6 (Hexamer coverage zero function)**.** Let t be a threshold $\in [0,1]$, T a set of thresholds we want to consider, h a hexamer and C the set of chromosomes we consider. We define:

- A_h as the average value of HC for the specified h and all chromosomes in C (that is, $\sum_{c\in C}\mathrm{HC}(c,h)/|C|$);

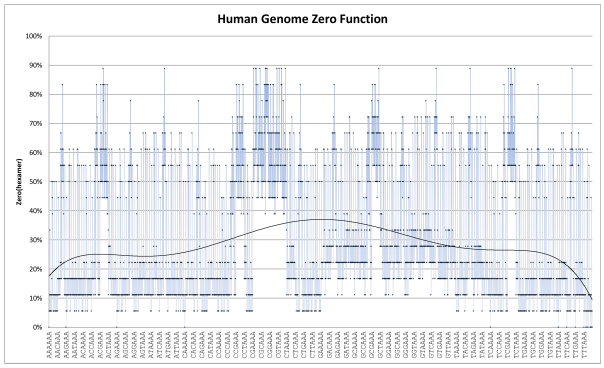


Figure 3. A plot of the hexamer coverage zero function (in blue) and of its 6^{th}-order fitting polynomial.

- $P_h(t) = \begin{cases} 1 & \text{if } A_h > t, \\ 0 & \text{otherwise.} \end{cases}$

The *hexamer coverage zero function* relative to a hexamer h and a set of thresholds T is defined as:

$$\mathrm{HCZ}(h) = 1 - \frac{1}{|T|}\sum_{k\in T} P_h(k)$$

Definition 7 (Row coverage zero function)**.** Let t be a threshold $\in [0,1]$ and let H be the set of all possible hexamers. Define:

$$\mathrm{RC}_{c,t} = \begin{cases} 1 & \text{if } \sum_{h\in H}\mathrm{HC}(c,h) > t, \\ 0 & \text{otherwise.} \end{cases}$$

The *row coverage zero function* relative to a chromosome c and a set of thresholds T is defined as:

$$\mathrm{RCZ}(c) = \frac{1}{|T|}\sum_{t\in T} \mathrm{RC}_{c,t}$$

A plot of the hexamer coverage zero function (considering the hexamers in their lexicographic order, see Figure 3) revealed several high peaks in the 80%/90% region (especially for the hexamers in the ACCGCG-ACGTCG region and for ATCGCG, the CCGACG-CGTTCG, GCGTCG region, GTCGCG, TACGCG, the TCGACG-TCGTCG region, TTCGCG). The trend is almost symmetric: a 6^{th}-order fitting polynomial shows a slow increase at first, a stationary region and a middle peak, followed by a second stationary region and a quick decrease in value.

We then transposed the hexamer coverage distribution matrix to get the *column coverage hexamer order* distribution; by splitting the resulting matrix by row (each row corresponds to a chromosome) and ordering it ascendingly by frequency we get the *column coverage value order* distribution, also called *Zipf elongated hexamers* distribution. The plot of the column coverage hexamer order distribution (Figure 4) reveals the following facts.

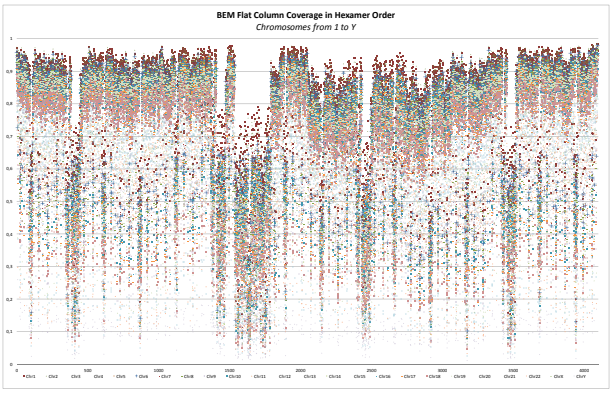


Figure 4. A plot of the column coverage hexamer order distribution.

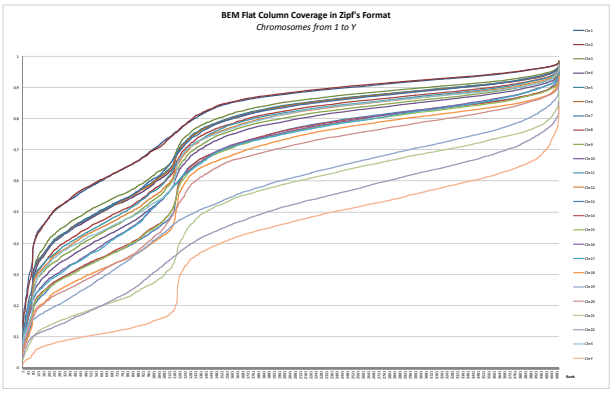


Figure 6. A plot of the Zipf elongated hexamers distribution.

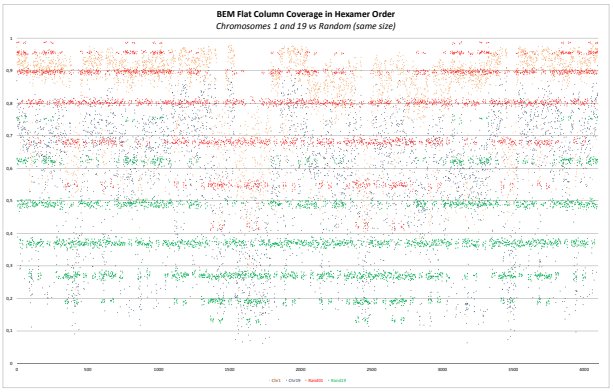


Figure 5. A plot of the column coverage hexamer order distribution for chromosomes 1, 19, Rand1 and Rand19.

- The coverage values for each chromosome are roughly independent from the hexamers, except in certain regions. In regular areas most coverage values tend to accumulate in the interval [0,3, 0,97], with a higher density in [0,7, 0,97].
- In some regions centered around the 400th, 1450th, 1500-1750th, 2450-2500th and 3500th hexamer the coverage values are shifted lower by about 0,35; chromosomes exhibiting high coverage values in the rest of the plot tend to present relatively high values in these areas as well.
- We also plotted the column coverage hexamer order distribution for chromosomes 1 and 19 and for Rand01 and Rand19 (Figure 5). A comparison of the plots revealed significant differences between them: while the 6^{th}-order fitting polynomials are similar in shape (except in the $[0, 500]$ region), the coverage values for the random chromosomes are lower than the ones for the real chromosomes and tend to gather around multiples of 0,1.
- The Zipf elongated hexamers distribution has not a linear plot (see Figure 6) — most chromosomes present a marked increase of about 0,15 units at around 1180. Chromosomes 1, 2, 19 and 22 present a smoother increase in value. The resulting graph at first seems comparable to the one of a sigmoid curve restricted to $x > 0$, though the sudden jump definitely precludes a good fitting. A comparison between chromosomes 1 and 19 and the random sequences Rand01 and Rand02 showed that the latter have a stair-like plot, with value increases occurring in the same positions, while the graphs for the real chromosomes do not present such marked variations.

We then obtained the *dictionary elongation matrices* by removing duplicate hexamers from the string elongation matrices and derived the *DEM hexamer count in hexamer order* matrix.

Definition 8 (DEM hexamer count in hexamer order matrix)**.** Let h be an hexamer, c a chromosome and DEMc the $n \times m$ dictionary elongation matrix for c. We define an indicator function $\delta(h, c, i, j)$ as follows:

$$\delta(h, c, i, j) = \begin{cases} 1 & \text{if DEM}c_{i,j} = h, \\ 0 & \text{otherwise.} \end{cases}$$

The coefficients of the *DEM hexamer count in hexamer order* matrix are defined as follows:

$$\text{DHC}(h, c) = \sum_{i=1}^{n} \sum_{j=1}^{m} \delta(h, c, i, j)$$

The resulting matrix was analyzed to obtain:

- the maximum, minimum and average hexamer count for each chromosome;
- the normalized maximum, minimum and average hexamer count;

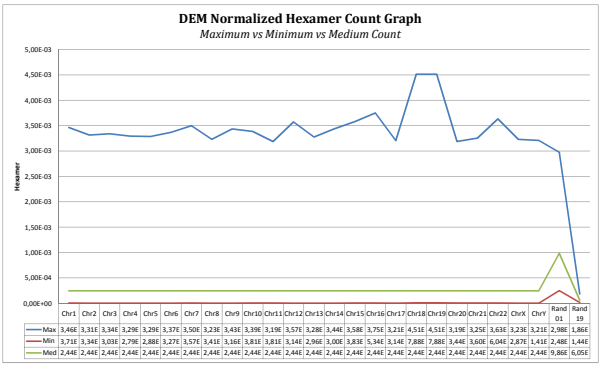


Figure 7. A plot of the DEM normalized hexamer count showing the maximum, minimum and average values.

- the decimal logarithm of the above mentioned statistical indicators;
- the hexamers with the minimum and maximum number of occurrences per chromosome.

The resulting findings were:

- the hexamer appearing most frequently in all chromosomes was AAAAAA, while the ones appearing less frequently were TACGCG, CGCGTA, CGATCG, TCGACG;
- a plot of the DEM normalized hexamer count (Figure 7) showed that the maximum count is approximately constant at $3{,}29 \times 10^{-3}$, with minor fluctuations, up to chromosome 13, then it presents four peaks at chromosomes 16, 18, 19 and 22; the minimum count has very strong fluctuations and presents peaks at the same locations as the maximum count;
- the X and Y chromosomes have a very low minimum hexamer count.

Furthermore, these facts do not hold at all for the random sequences Rand01 and Rand19.

5 Conclusions

We analyzed the human genome from an informational point of view by extracting indicators we deemed significant, plotting elongation matrices for each chromosome and deriving coverage distributions from them. The data we obtained is noteworthy and deserves additional study; in particular, the peak of the hexamer coverage zero function, the drop in the column coverage hexamer order distribution plot and the trend of the Zipf elongated hexamer distribution strongly suggest that the relevant hexamers have a well-defined meaning within the genome. A possible research direction and expansion of our work could be searching for those meanings and explaining them in a biological/biochemical context, linking our findings with well-specified reactions and cellular phenomena.

References

[1] *Apache™ Hadoop®*. Available at http://hadoop.apache.org/.

[2] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: (2004).

[3] UCSC Genome Bioinformatics Group. *UCSC Genome Browser*. Available at http://hgdownload.cse.ucsc.edu/.

[4] Zhang Lab. *FASTA format*. Available at http://zhanglab.ccmb.med.umich.edu/FASTA/.

[5] Donald Miner and Adam Shook. *Mapreduce Design Patterns Building Effective Algorithms and Analytics for Hadoop and Other Systems*. Oreilly & Associates Inc, 2012. ISBN: 9781449327170.

[6] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge: Cambridge University Press, 2005. ISBN: 9780521835404.

[7] *NIST/SEMATECH e-Handbook of Statistical Methods*. Available at http://www.itl.nist.gov/div898/handbook/. Oct. 2013.

[8] Tom White. *Hadoop - The Definitive Guide: Storage and Analysis at Internet Scale (3. ed., revised and updated)*. O'Reilly, 2012. ISBN: 9781449311520.

Appendix: Detailed plots and comments

In this Appendix we provide higher-sized, commented versions of the plots we derived during our work.

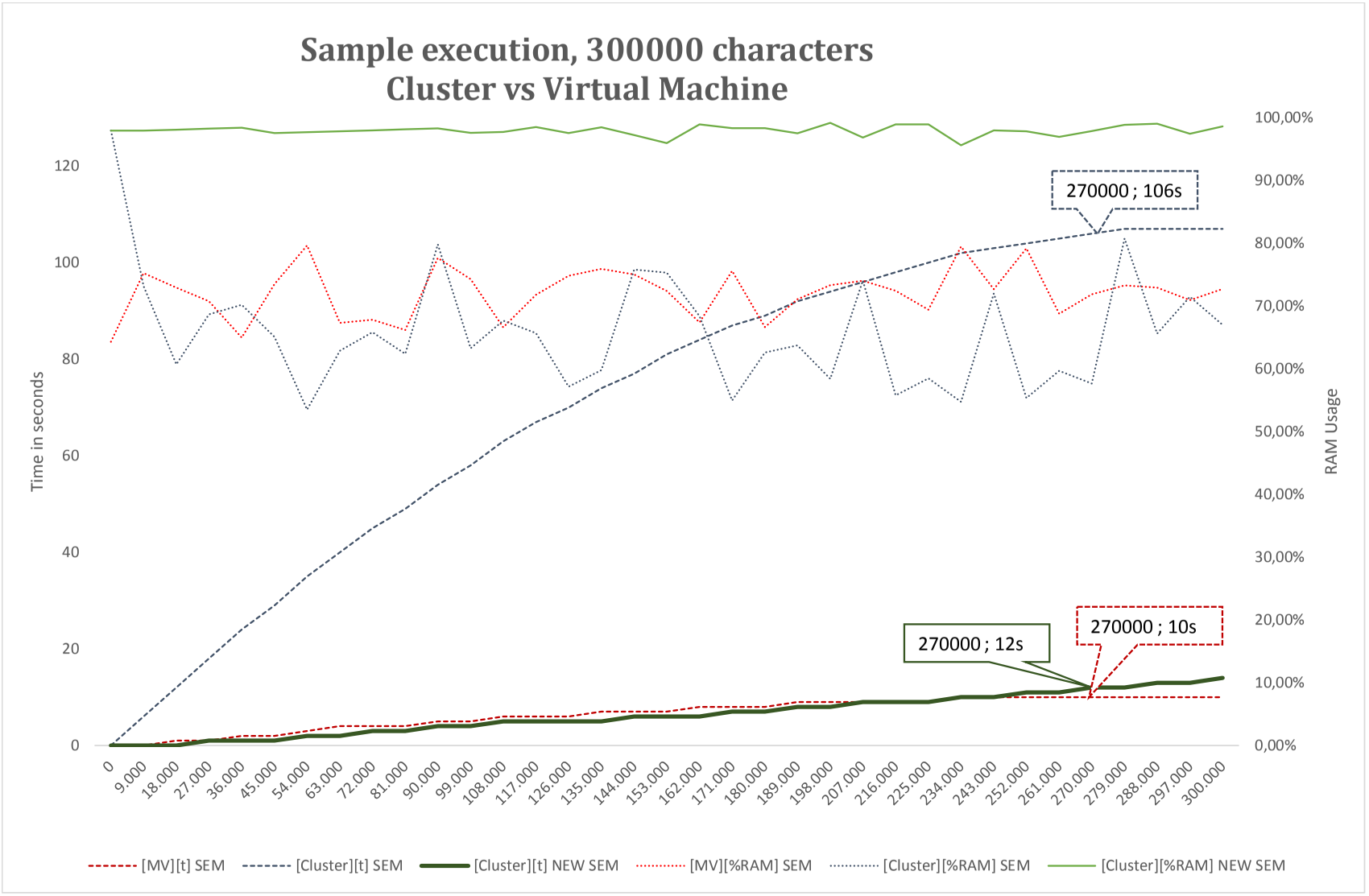


Figure 8. A comparison of the execution time and RAM usage of the old and the new software on a sample of 300000 bases. Note the exceptionally high difference between the running time of the old code in our VM environment and on the cluster, the spikes in RAM usage caused by the old architecture — no matter what the environment — and the stable trends of the second version of our software.

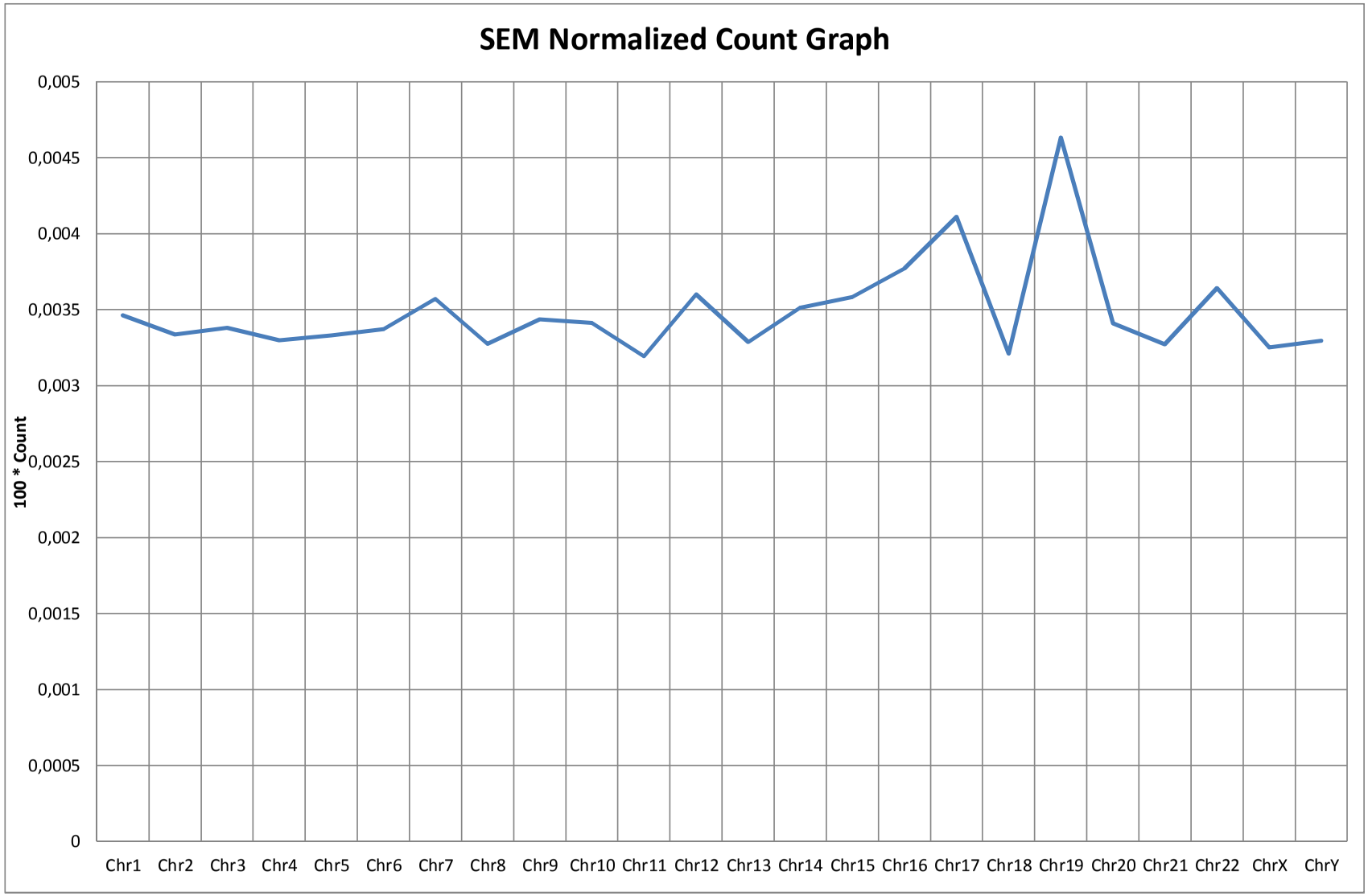


Figure 9. A plot of the SEM normalized count distribution. The trend is (more or less) stationary; peaks can be observed for chromosomes 17 and 19.

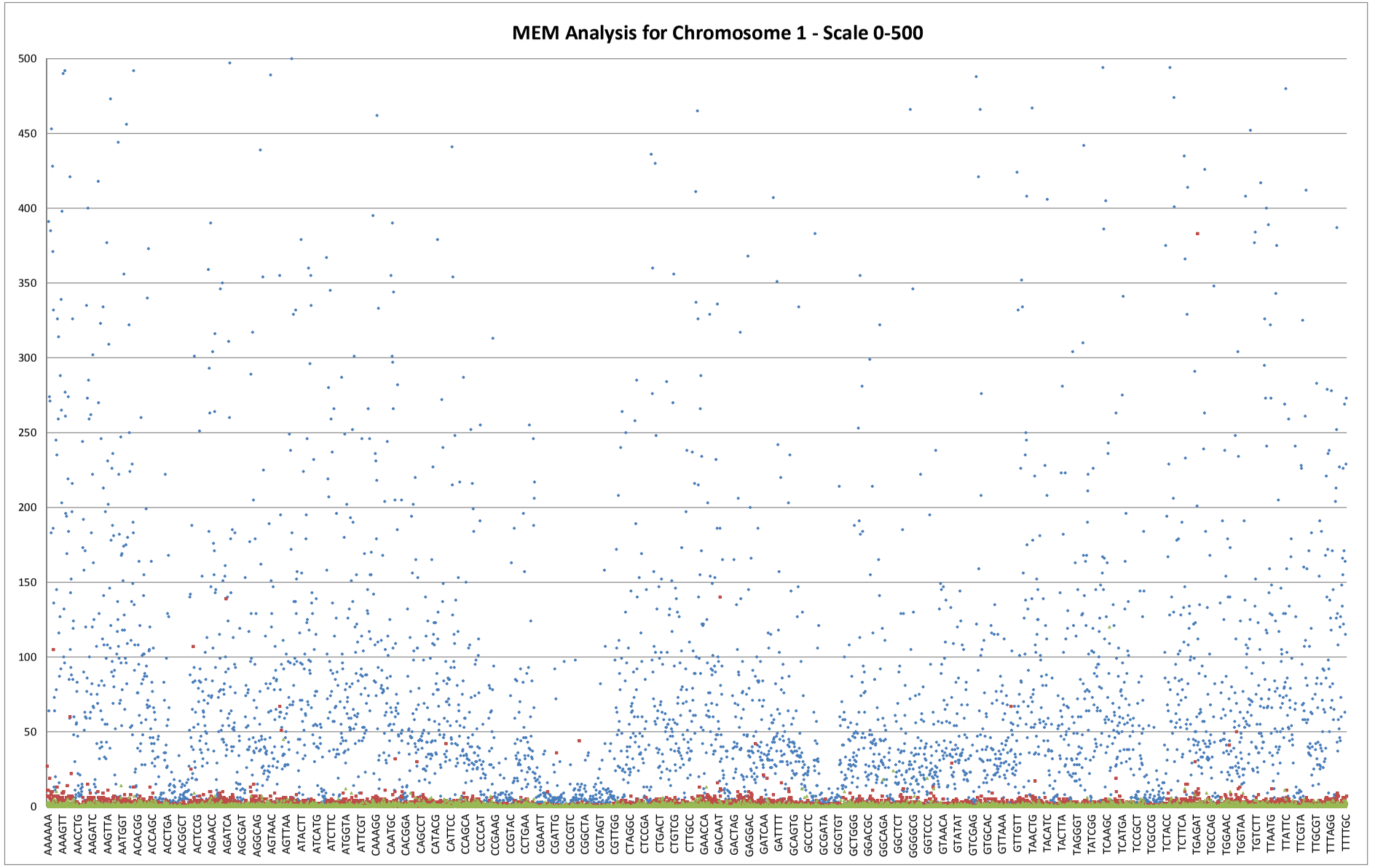


Figure 10. Plot of the MEM for chromosome 1. All the possible hexamers are on the x axis (in lexicographic order), while their respective maximum, minimum and average values are on the y axis. The ordinates are limited to the interval $[0,500]$. The maximum values are in blue and are extremely variable, ranging even up to 500, while the average and minimum values (in red and green) are generally less than 50.

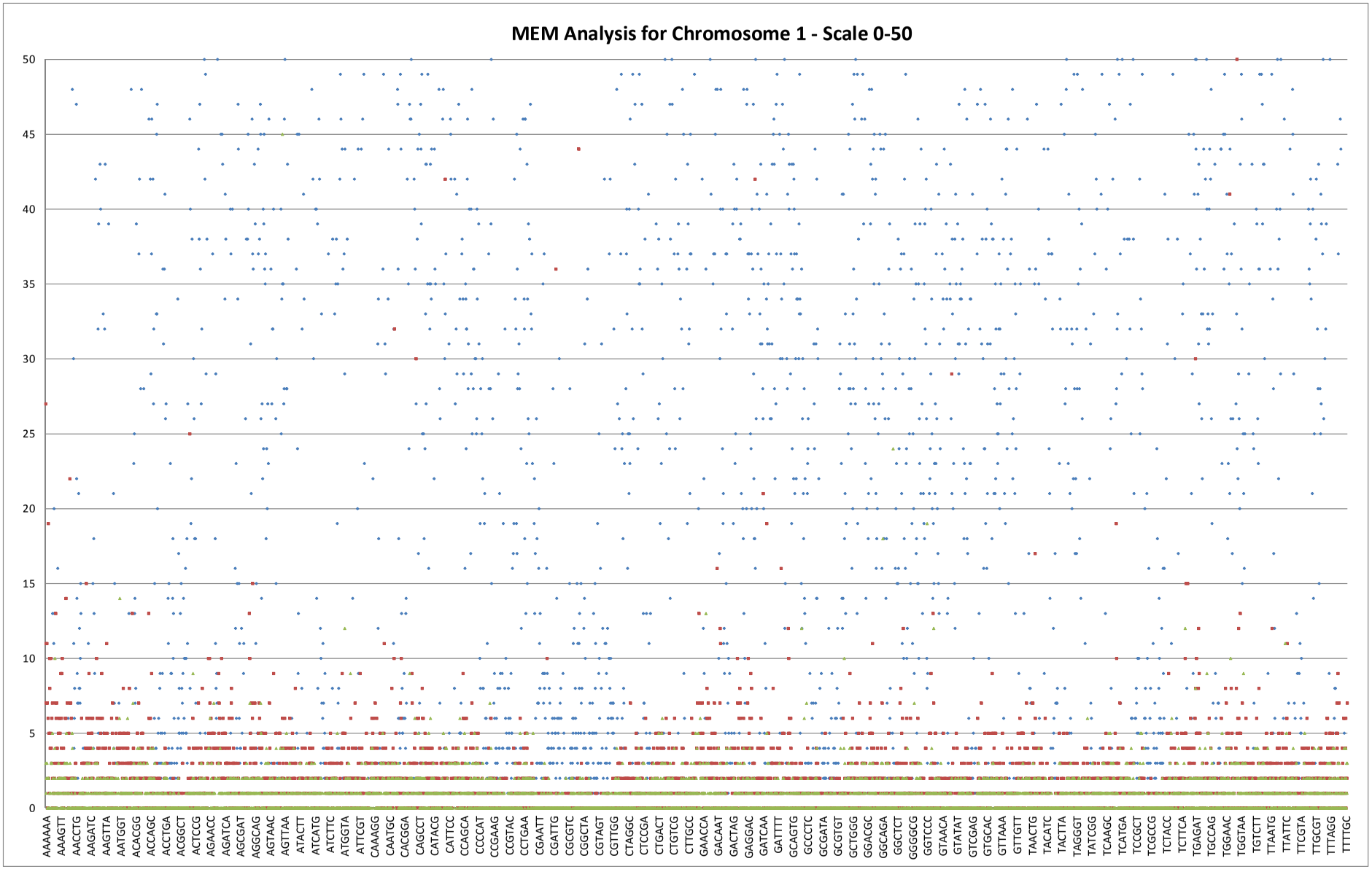


Figure 11. Plot of the MEM for chromosome 1; the ordinates are limited to the interval $[0,50]$. The distribution of the minimum and average values is more evident: most minima are equal to 0, 1 or 2, while average values are generally located in the interval $[1,7]$.

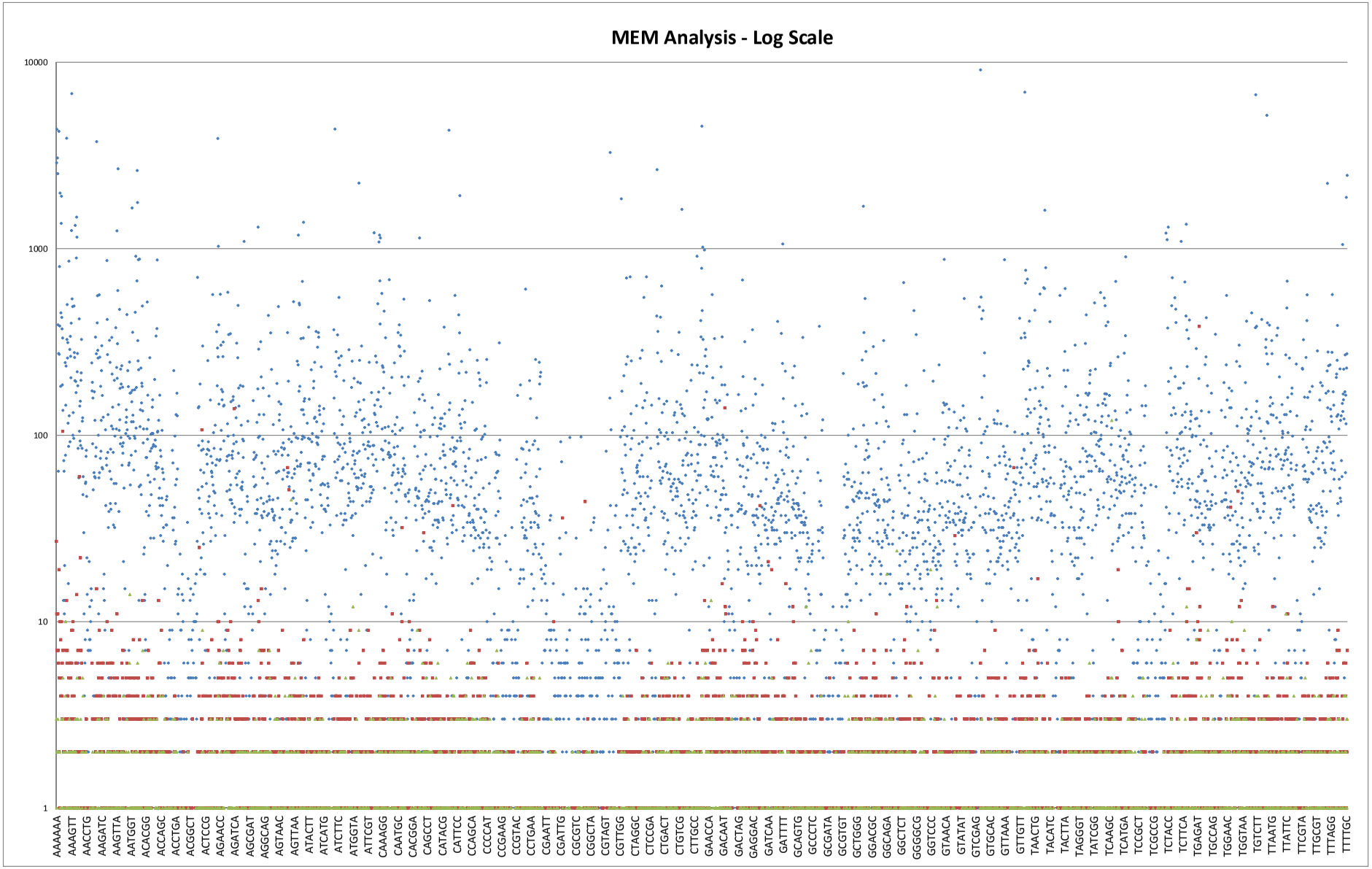


Figure 12. Plot of the MEM for chromosome 1; the y axis uses a logarithmic scale. This graph shows how most maximum values are located in the [10, 1000] interval, with some exceptions (namely, the regions centered around the hexamers ACGAGA, CCGATC, CGAGAA-CGTGCA, GCGCAG, TCGCCT, where minima reach values as low as 1).

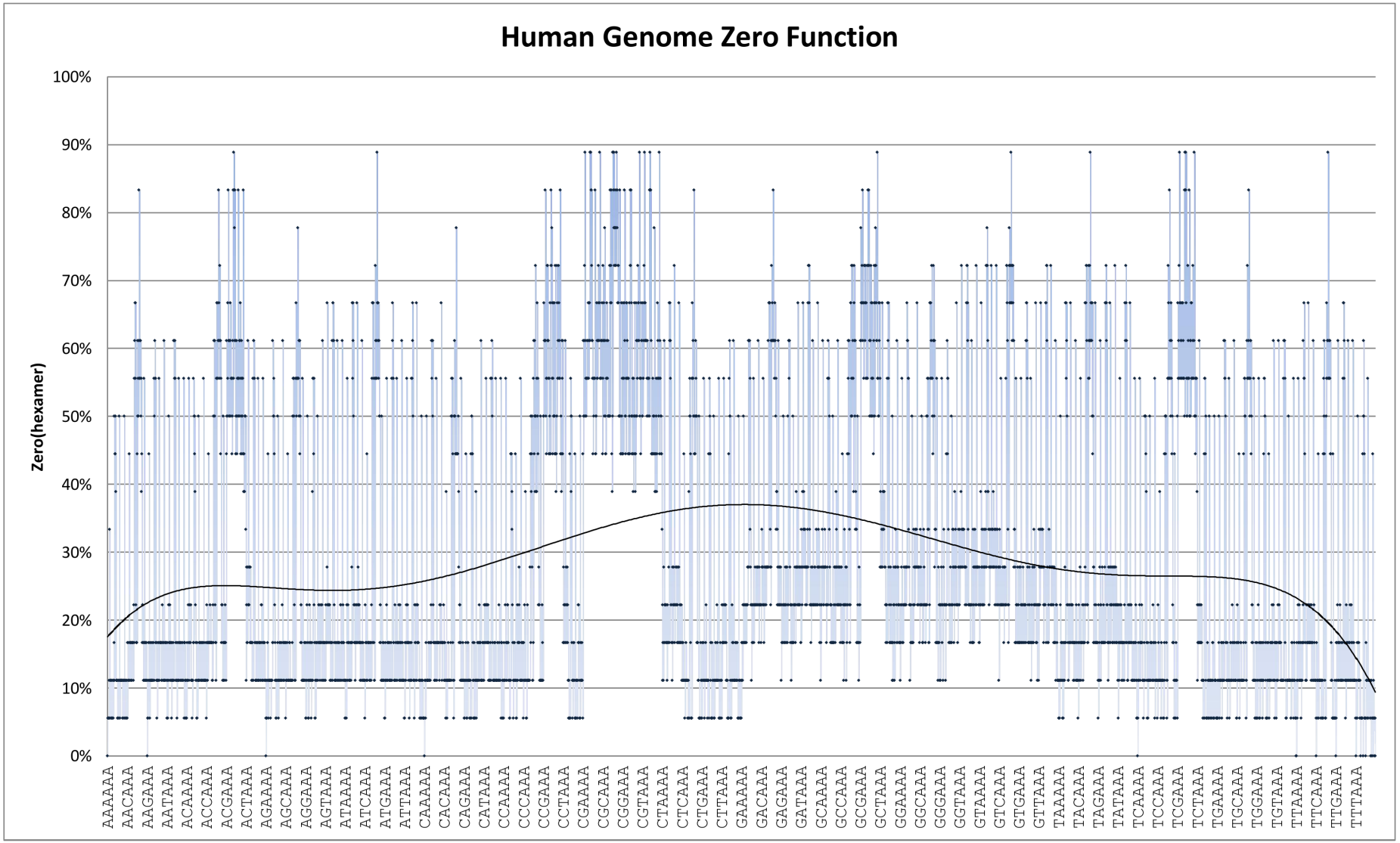


Figure 13. The hexamer coverage zero function (in blue) and its 6th order fitting polynomial (in black). The zero function is extremely variable; most values lie in the 5%-70% region, with some noticeable "peaks" characterized by high values (in the 45%-90% region) comprising the ACCGCG-ACGTCG, CCGACG-CGTTCG, TCGACG-TCGTCG regions and other minor ones. The fitting polynomial has the equation $y = -1 \times 10^{-20}x^6 + 2 \times 10^{-16}x^5 - 7 \times 10^{-13}x^4 + 2 \times 10^{-9}x^3 - 1 \times 10^{-6}x^2 + 0{,}0005x + 0{,}1746$; the R^2 coefficient is equal to 0,0676, which is pretty low.

Zero function threshold value > 85%																	
ACGCGA	ATCGCG	CGAACG	CGACCG	CGACGA	CGACGC	CGATCG	CGCGAA	CGCGAC	CGCGAT	CGCGTA	CGGTCG	CGTACG	CGTCGA	CGTTCG	GCGTCG	GTCGCG	TACGCG
88,89%	88,89%	88,89%	88,89%	88,89%	88,89%	88,89%	88,89%	88,89%	88,89%	88,89%	88,89%	88,89%	88,89%	88,89%	88,89%	88,89%	88,89%
TCGACG	TCGCGA	TCGCGT	TCGTCG	TTCGCG													
88,89%	88,89%	88,89%	88,89%	88,89%													
Zero function threshold value > 80%																	
AACGCG	ACCGCG	ACGACG	ACGCCG	ACGCGA	ACGCGC	ACGCGT	ACGGCG	ACGTCG	ATCGCG	CCGACG	CCGCGA	CCGTCG	CGAACG	CGACCG	CGACGA	CGACGC	CGACGG
83,33%	83,33%	83,33%	83,33%	88,89%	83,33%	83,33%	83,33%	83,33%	88,89%	83,33%	83,33%	83,33%	88,89%	88,89%	88,89%	88,89%	83,33%
CGACGT	CGAGCG	CGATCG	CGCCGA	CGCCGT	CGCGAA	CGCGAC	CGCGAG	CGCGAT	CGCGGA	CGCGTA	CGCGTC	CGCGTT	CGCTCG	CGGACG	CGGCGA	CGGCGT	CGGTCG
83,33%	83,33%	88,89%	83,33%	83,33%	88,89%	88,89%	83,33%	88,89%	83,33%	88,89%	83,33%	83,33%	83,33%	83,33%	83,33%	83,33%	88,89%
CGTACG	CGTCCG	CGTCGA	CGTCGC	CGTCGG	CGTCGT	CGTTCG	CTCGCG	GACGCG	GCGACG	GCGCGA	GCGCGT	GCGTCG	GTCGCG	TACGCG	TCCGCG	TCGACG	TCGCCG
88,89%	83,33%	88,89%	83,33%	83,33%	83,33%	88,89%	83,33%	83,33%	83,33%	83,33%	83,33%	88,89%	88,89%	88,89%	83,33%	88,89%	83,33%
TCGCGA	TCGCGC	TCGCGG	TCGCGT	TCGGCG	TCGTCG	TGCGCG	TTCGCG										
88,89%	83,33%	83,33%	88,89%	83,33%	88,89%	83,33%	88,89%										
Zero function threshold value > 75%																	
AACGCG	ACCGCG	ACGACG	ACGCCG	ACGCGA	ACGCGC	ACGCGG	ACGCGT	ACGGCG	ACGTCG	AGCGCG	ATCGCG	CACGCG	CCGACG	CCGCGA	CCGCGT	CCGTCG	CGAACG
83,33%	83,33%	83,33%	83,33%	88,89%	83,33%	77,78%	83,33%	83,33%	83,33%	77,78%	88,89%	77,78%	83,33%	83,33%	77,78%	83,33%	88,89%
CGACCG	CGACGA	CGACGC	CGACGG	CGACGT	CGAGCG	CGATCG	CGCACG	CGCCGA	CGCCGT	CGCGAA	CGCGAC	CGCGAG	CGCGAT	CGCGCA	CGCGCG	CGCGCT	CGCGGA
88,89%	88,89%	88,89%	83,33%	83,33%	83,33%	88,89%	77,78%	83,33%	83,33%	88,89%	88,89%	83,33%	88,89%	77,78%	77,78%	77,78%	83,33%
CGCGGT	CGCGTA	CGCGTC	CGCGTG	CGCGTT	CGCTCG	CGGACG	CGGCGA	CGGCGT	CGGTCG	CGTACG	CGTCCG	CGTCGA	CGTCGC	CGTCGG	CGTCGT	CGTGCG	CGTTCG
77,78%	88,89%	83,33%	77,78%	83,33%	83,33%	83,33%	83,33%	83,33%	88,89%	88,89%	83,33%	88,89%	83,33%	83,33%	83,33%	77,78%	88,89%
CTCGCG	GACGCG	GCGAAC	GCGACG	GCGCGA	GCGCGT	GCGTAC	GCGTCG	GTACGC	GTCGAC	GTCGCG	TACGCG	TCCGCG	TCGACG	TCGCCG	TCGCGA	TCGCGC	TCGCGG
83,33%	83,33%	77,78%	83,33%	83,33%	83,33%	77,78%	88,89%	77,78%	77,78%	88,89%	88,89%	83,33%	88,89%	83,33%	88,89%	83,33%	83,33%
TCGCGT	TCGGCG	TCGTCG	TGCGCG	TTCGCG													
88,89%	83,33%	88,89%	83,33%	88,89%													
Zero function threshold value > 70%																	
AACGCG	ACCGCG	ACCGGT	ACGACG	ACGCCG	ACGCGA	ACGCGC	ACGCGG	ACGCGT	ACGGCG	ACGTCG	AGCGCG	ATCGAC	ATCGCG	CACGCG	CCCGCG	CCGACG	CCGATA
83,33%	83,33%	72,22%	83,33%	83,33%	88,89%	83,33%	77,78%	83,33%	83,33%	83,33%	77,78%	72,22%	88,89%	77,78%	72,22%	83,33%	72,22%
CCGATC	CCGCGA	CCGCGC	CCGCGG	CCGCGT	CCGGCG	CCGGTA	CCGTAC	CCGTCG	CGAACC	CGAACG	CGACCG	CGACGA	CGACGC	CGACGG	CGACGT	CGACTA	CGAGCG
72,22%	83,33%	72,22%	72,22%	77,78%	72,22%	72,22%	72,22%	83,33%	72,22%	88,89%	88,89%	88,89%	88,89%	83,33%	83,33%	72,22%	83,33%
CGATAC	CGATCG	CGCAAC	CGCACG	CGCCCG	CGCCGA	CGCCGG	CGCCGT	CGCGAA	CGCGAC	CGCGAG	CGCGAT	CGCGCA	CGCGCC	CGCGCG	CGCGCT	CGCGGA	CGCGGC
72,22%	88,89%	72,22%	77,78%	72,22%	83,33%	72,22%	83,33%	88,89%	88,89%	83,33%	88,89%	77,78%	72,22%	77,78%	77,78%	83,33%	72,22%
CGCGGG	CGCGGT	CGCGTA	CGCGTC	CGCGTG	CGCGTT	CGCTAC	CGCTAG	CGCTCG	CGGACG	CGGCGA	CGGCGC	CGGCGG	CGGCGT	CGGTAC	CGGTCG	CGTACC	CGTACG
72,22%	77,78%	88,89%	83,33%	77,78%	83,33%	72,22%	72,22%	83,33%	83,33%	83,33%	72,22%	72,22%	83,33%	72,22%	88,89%	72,22%	88,89%
CGTCCG	CGTCGA	CGTCGC	CGTCGG	CGTCGT	CGTGCG	CGTTCG	CTAGCG	CTCGCG	GACGAC	GACGCG	GATCGA	GATCGC	GATCGG	GCCGAT	GCCGCG	GCCGTA	GCGAAC
83,33%	88,89%	83,33%	83,33%	83,33%	77,78%	88,89%	72,22%	83,33%	72,22%	83,33%	72,22%	72,22%	72,22%	72,22%	72,22%	72,22%	77,78%
GCGACC	GCGACG	GCGATA	GCGATC	GCGCAA	GCGCCG	GCGCGA	GCGCGC	GCGCGG	GCGCGT	GCGCTA	GCGGAC	GCGGTA	GCGTAA	GCGTAC	GCGTAG	GCGTCG	GGCGCG
72,22%	83,33%	72,22%	72,22%	72,22%	72,22%	83,33%	72,22%	72,22%	83,33%	72,22%	72,22%	72,22%	72,22%	77,78%	72,22%	88,89%	72,22%
GGCGTA	GGTACG	GGTCGA	GGTCGC	GGTTCG	GTACCG	GTACGA	GTACGC	GTACGG	GTAGCG	GTATCG	GTCCGA	GTCCGC	GTCGAA	GTCGAC	GTCGAT	GTCGCA	GTCGCG
72,22%	72,22%	72,22%	72,22%	72,22%	72,22%	72,22%	77,78%	72,22%	72,22%	72,22%	72,22%	72,22%	72,22%	77,78%	72,22%	72,22%	88,89%
GTCGGA	GTCGGT	GTCGTA	GTTCGC	GTTCGG	GTTGCG	TACCGC	TACCGG	TACGAC	TACGCC	TACGCG	TAGCGC	TAGTCG	TATCGC	TCCGAC	TCCGCG	TCGACC	TCGACG
72,22%	72,22%	72,22%	72,22%	72,22%	72,22%	72,22%	72,22%	72,22%	72,22%	88,89%	72,22%	72,22%	72,22%	72,22%	83,33%	72,22%	88,89%
TCGCCG	TCGCGA	TCGCGC	TCGCGG	TCGCGT	TCGGAC	TCGGCG	TCGTAC	TCGTCG	TGCGAC	TGCGCG	TTCGCG						
83,33%	88,89%	83,33%	83,33%	88,89%	72,22%	83,33%	72,22%	88,89%	72,22%	83,33%	88,89%						

Table 2. A list of the hexamers having significantly high or low zero function values.

Zero function threshold value < 10%																		
AAAAAA	AAAAAC	AAAAAG	AAAAAT	AAAACA	AAAACT	AAAAGA	AAAATA	AAAATG	AAAATT	AAACAA	AAACAT	AAAGAA	AAAGAG	AAAGCA	AAAGGA	AAATAA	AAATAT
0,00%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%
AAATGA	AAATGT	AAATTA	AAATTT	AACAAA	AAGAAA	AAGAAG	AAGAAT	AAGAGA	AAGGAA	AATAAA	AATAAT	AATATT	AATGAA	AATTCT	AATTTA	AATTTT	ACAAAA
5,56%	5,56%	5,56%	5,56%	5,56%	0,00%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%
ACACAG	ACAGAA	ACAGAG	ACATTT	ACTTTT	AGAAAA	AGAAAG	AGAAAT	AGAAGA	AGAATT	AGAGAA	AGAGAG	AGAGGA	AGATTT	AGGAAA	AGGAAG	AGGAGA	AGGAGG
5,56%	5,56%	5,56%	5,56%	5,56%	0,00%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%
AGGCAG	AGGCTG	AGGGAA	AGTTTT	ATAAAA	ATAAAT	ATATTT	ATCTTT	ATGAAA	ATGTTT	ATTAAA	ATTATT	ATTCTT	ATTTAA	ATTTAT	ATTTCA	ATTTCT	ATTTGT
5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%
ATTTTA	ATTTTC	ATTTTG	ATTTTT	CAAAAA	CAAAAT	CACACA	CACAGA	CACCTG	CAGAAA	CAGAAG	CAGAGA	CAGAGG	CAGCAG	CAGCCA	CAGCCT	CAGCTG	CAGGAA
5,56%	5,56%	5,56%	0,00%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%
CAGGAG	CAGGCA	CAGGGA	CATTCT	CATTTA	CATTTC	CATTTT	CCACAG	CCACCA	CCACTG	CCAGGA	CCAGGC	CCATTT	CCCAGC	CCCAGG	CCTCAG	CCTCCA	CCTCCC
5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%
CCTCCT	CCTCTG	CCTGCC	CCTGCT	CCTGGA	CCTGGG	CCTGTG	CCTTCC	CCTTCT	CCTTTT	CTCAGC	CTCCAG	CTCCCA	CTCCCT	CTCCTC	CTCCTG	CTCCTT	CTCTCT
5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%
CTCTGA	CTCTGC	CTCTGG	CTCTGT	CTCTTT	CTGAAA	CTGAGA	CTGCAG	CTGCCT	CTGCTG	CTGCTT	CTGGAA	CTGGAG	CTGGCT	CTGGGA	CTGTCT	CTGTGA	CTGTGG
5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%
CTGTGT	CTGTTT	CTTCAG	CTTCCA	CTTCCC	CTTCCT	CTTCTC	CTTCTG	CTTCTT	CTTTCA	CTTTCC	CTTTCT	CTTTGT	CTTTTA	CTTTTC	CTTTTT	TAAAAA	TAAAAT
5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%
TAAATA	TAAATT	TAATTT	TAGAAA	TATAAA	TATTTA	TATTTC	TATTTT	TCAAAA	TCACAG	TCACTG	TCAGAA	TCATCT	TCATTT	TCCAGG	TCCCAG	TCCCCA	TCCCTG
5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	0,00%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%
TCCTCA	TCCTCT	TCCTGG	TCCTTT	TCTCAG	TCTCCA	TCTCCT	TCTCTC	TCTCTG	TCTCTT	TCTGAA	TCTGCA	TCTGCT	TCTGGG	TCTGTG	TCTGTT	TCTTCA	TCTTCC
5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%
TCTTCT	TCTTTC	TCTTTG	TCTTTT	TGAAAA	TGAAAG	TGAAAT	TGAAGA	TGAATT	TGAGAA	TGAGGA	TGATTT	TGCAAA	TGCAGA	TGCCTG	TGCTGG	TGCTGT	TGCTTT
5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%
TGGAAA	TGGAAG	TGGAGA	TGGAGG	TGGCAG	TGGCTG	TGGCTT	TGGGAA	TGGGAG	TGGGGA	TGGTGG	TGGTTT	TGTCTG	TGTCTT	TGTGAA	TGTGTG	TGTGTT	TGTTCT
5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%
TGTTTA	TGTTTC	TGTTTG	TGTTTT	TTAAAA	TTAAAT	TTAATT	TTATTT	TTCAAA	TTCACA	TTCAGA	TTCATT	TTCCAG	TTCCCA	TTCCCT	TTCCTC	TTCCTG	TTCCTT
5,56%	5,56%	5,56%	0,00%	0,00%	5,56%	5,56%	0,00%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%
TTCTCA	TTCTCC	TTCTCT	TTCTGA	TTCTGC	TTCTGG	TTCTGT	TTCTTC	TTCTTG	TTCTTT	TTGAAA	TTGCTT	TTGTTT	TTTAAA	TTTAAT	TTTACA	TTTATA	TTTATT
5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	0,00%	5,56%	5,56%	5,56%	0,00%	5,56%	5,56%	5,56%	0,00%
TTTCAA	TTTCAC	TTTCAG	TTTCAT	TTTCCA	TTTCCC	TTTCCT	TTTCTA	TTTCTC	TTTCTG	TTTCTT	TTTGAA	TTTGCA	TTTGCT	TTTGGA	TTTGTG	TTTGTT	TTTTAA
5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	0,00%	5,56%	5,56%	5,56%	0,00%	5,56%	5,56%	5,56%	5,56%	5,56%	5,56%	0,00%
TTTTAT	TTTTCA	TTTTCC	TTTTCT	TTTTGA	TTTTGG	TTTTGT	TTTTTA	TTTTTC	TTTTTG	TTTTTT							
0,00%	5,56%	5,56%	0,00%	5,56%	5,56%	5,56%	0,00%	5,56%	5,56%	0,00%							
Zero function threshold value < 5%																	
AAAAAA	AAGAAA	AGAAAA	ATTTTT	TATTTT	TGTTTT	TTAAAA	TTATTT	TTCTTT	TTTAAA	TTTATT	TTTCCT	TTTCTT	TTTTAA	TTTTAT	TTTTCT	TTTTTA	TTTTTT
0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%

Table 3. A list of the hexamers having significantly high or low zero function values (continued).

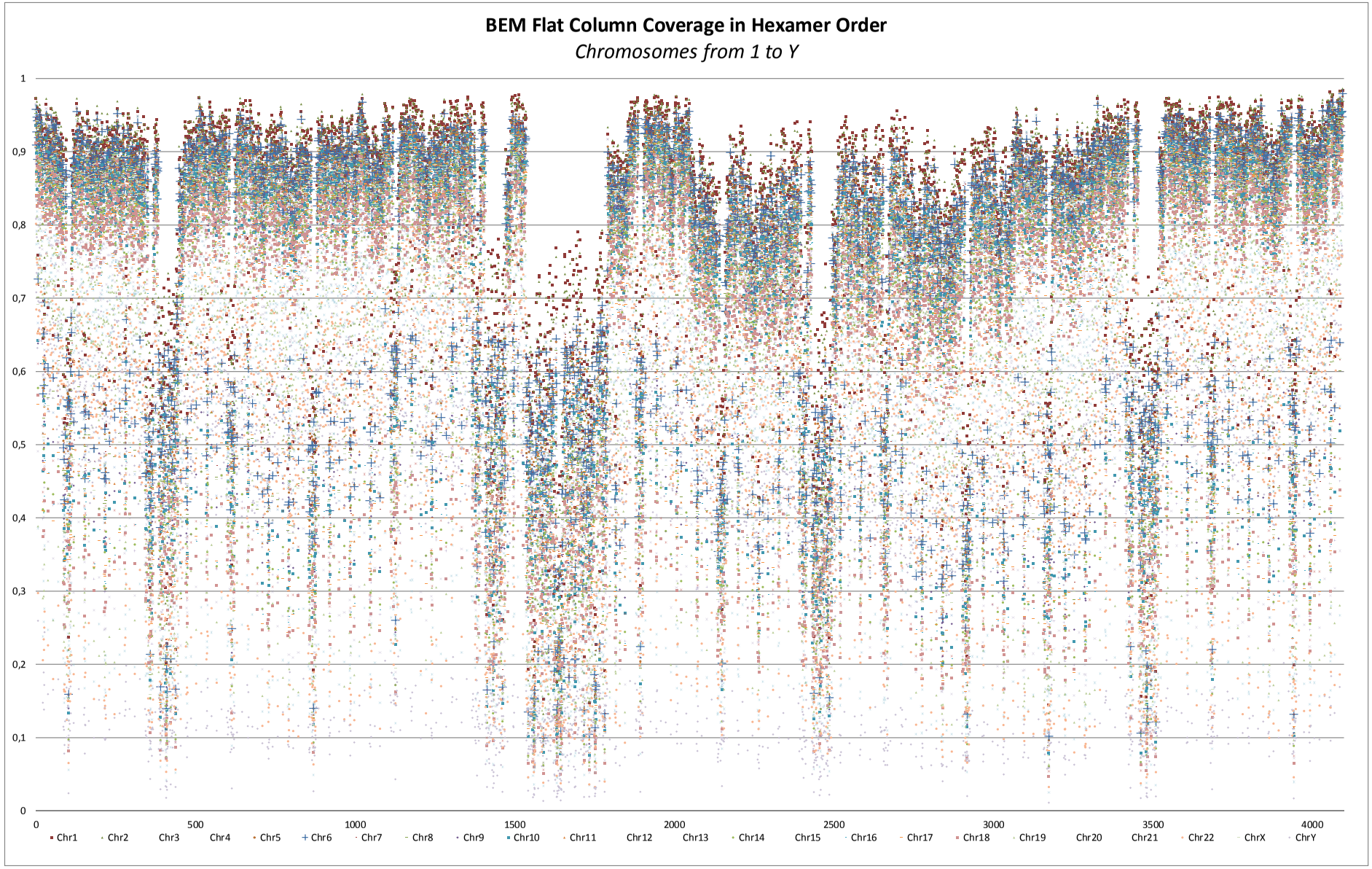


Figure 14. A plot of the BEM column coverage in hexamer order distribution. All the hexamers are (in their lexicographic order) on the x axis, the values assumed by the distribution function are on the y axis. Different chromosomes are represented by different colors and point types. Note how the column coverage values are roughly independent from the number of hexamers and the regions characterized by "low peaks".

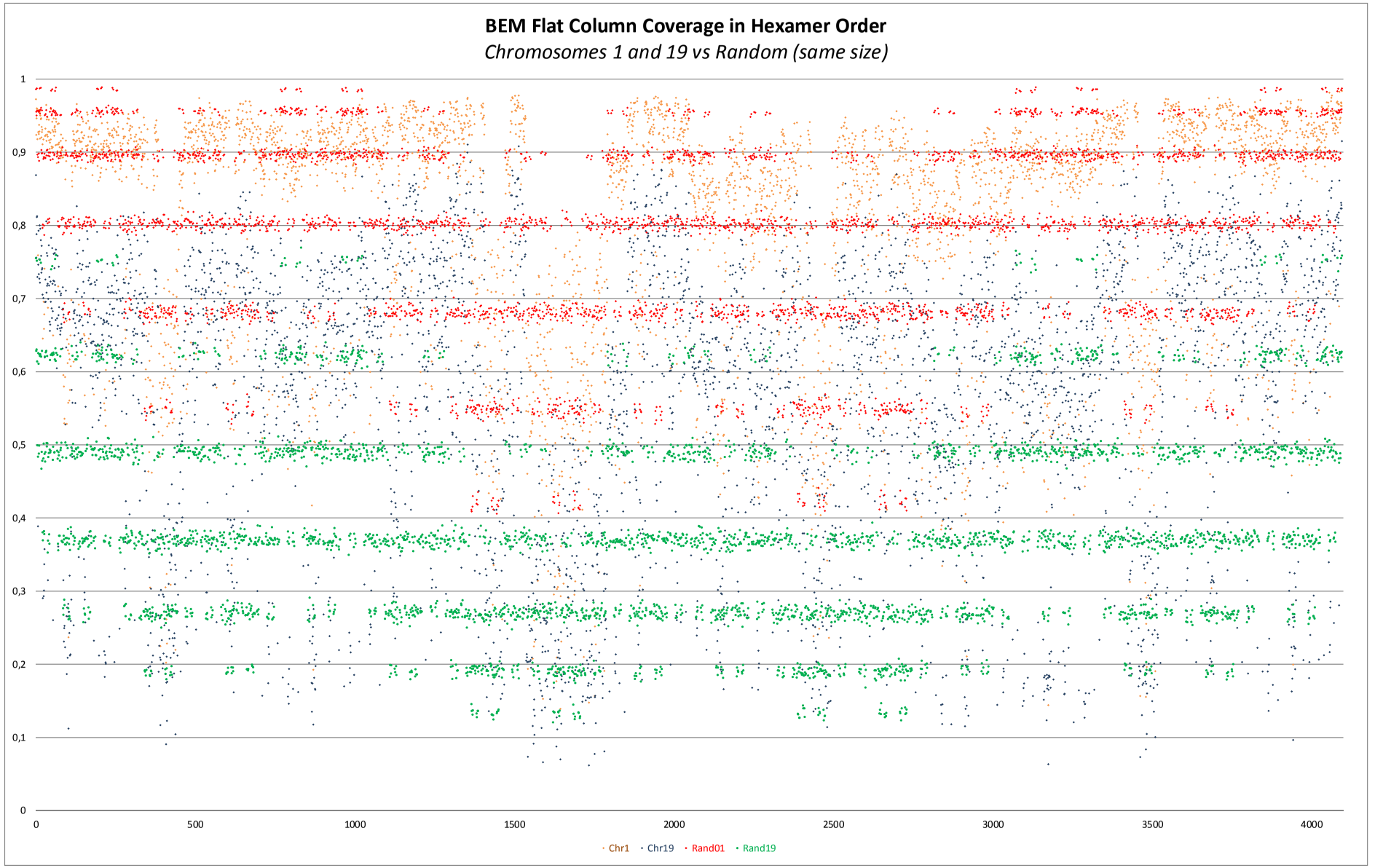


Figure 15. A comparison between the BEM column coverage in hexamer order distributions for chromosomes 1 and 19 and for the random sequences Rand1 and Rand19. The latter tend to form "bands", while the former are much more sparse and present a marked decrease in values in a region centered at approximately 1600.

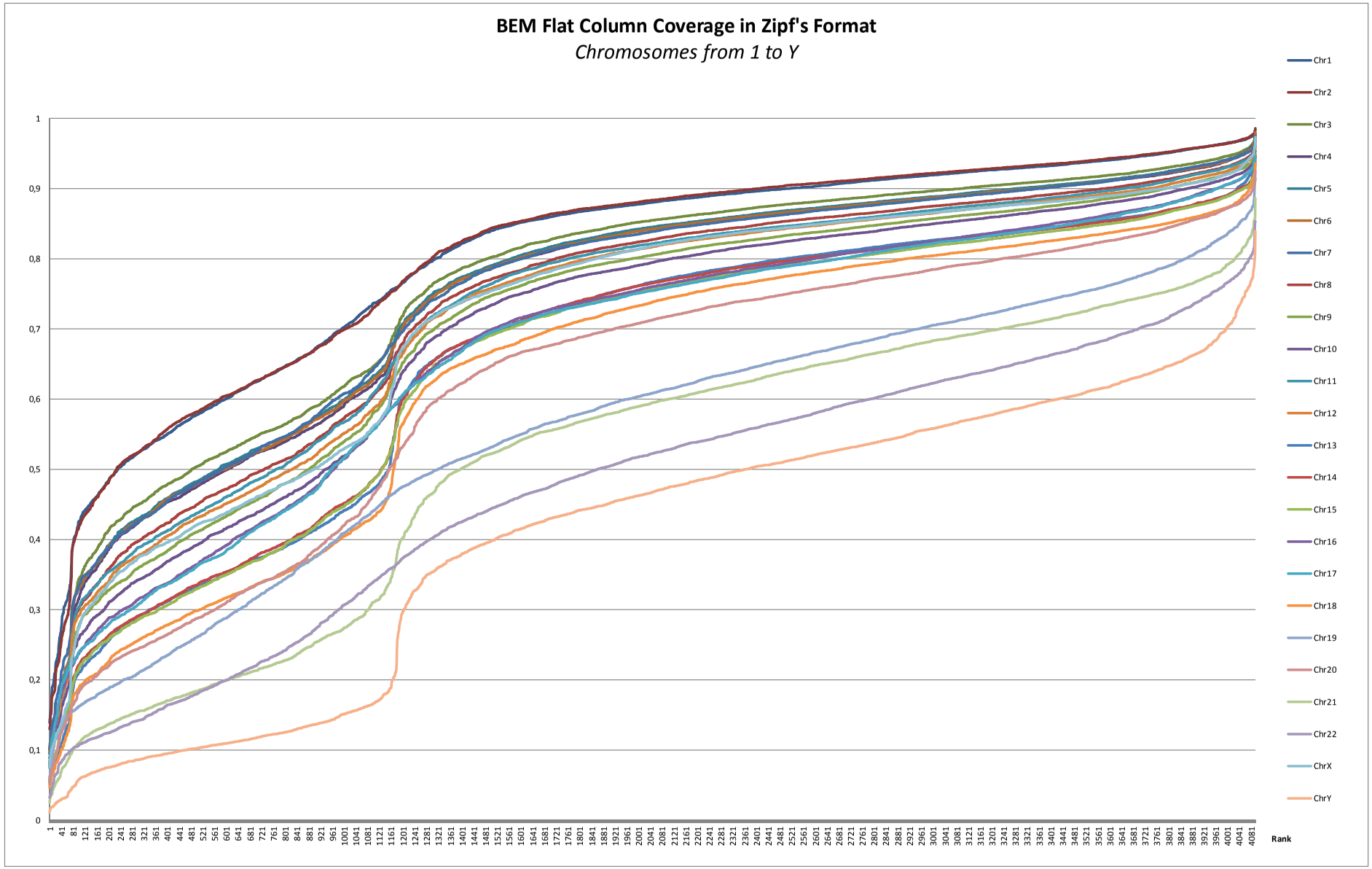


Figure 16. The Zipf elongated hexamers distribution. The plot is similar to a sigmoid curve and all lines have an inflection point at around 1180. Some chromosomes (e.g. 1, 2, 19 and 22) present a smoother increase in value.

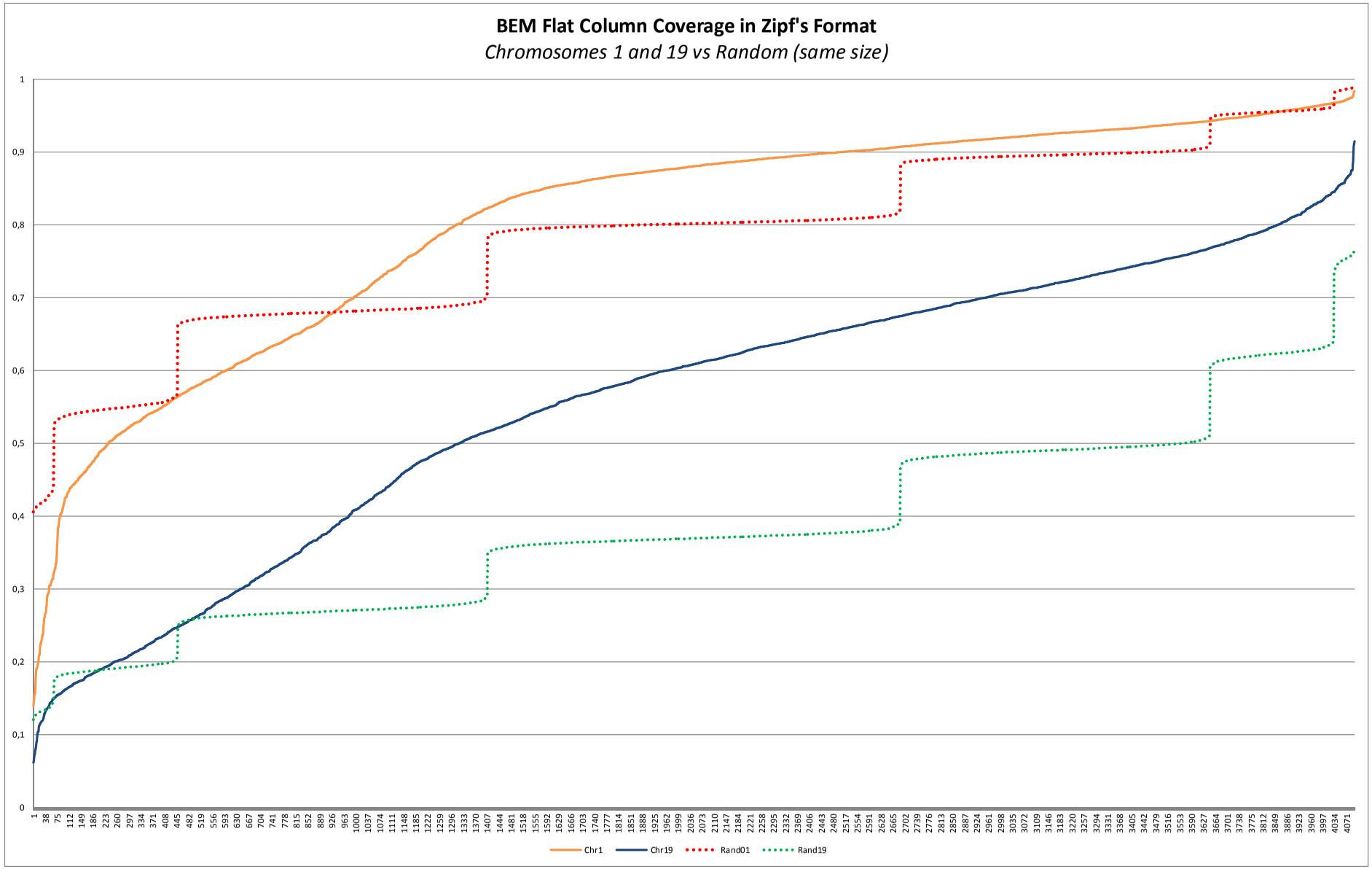
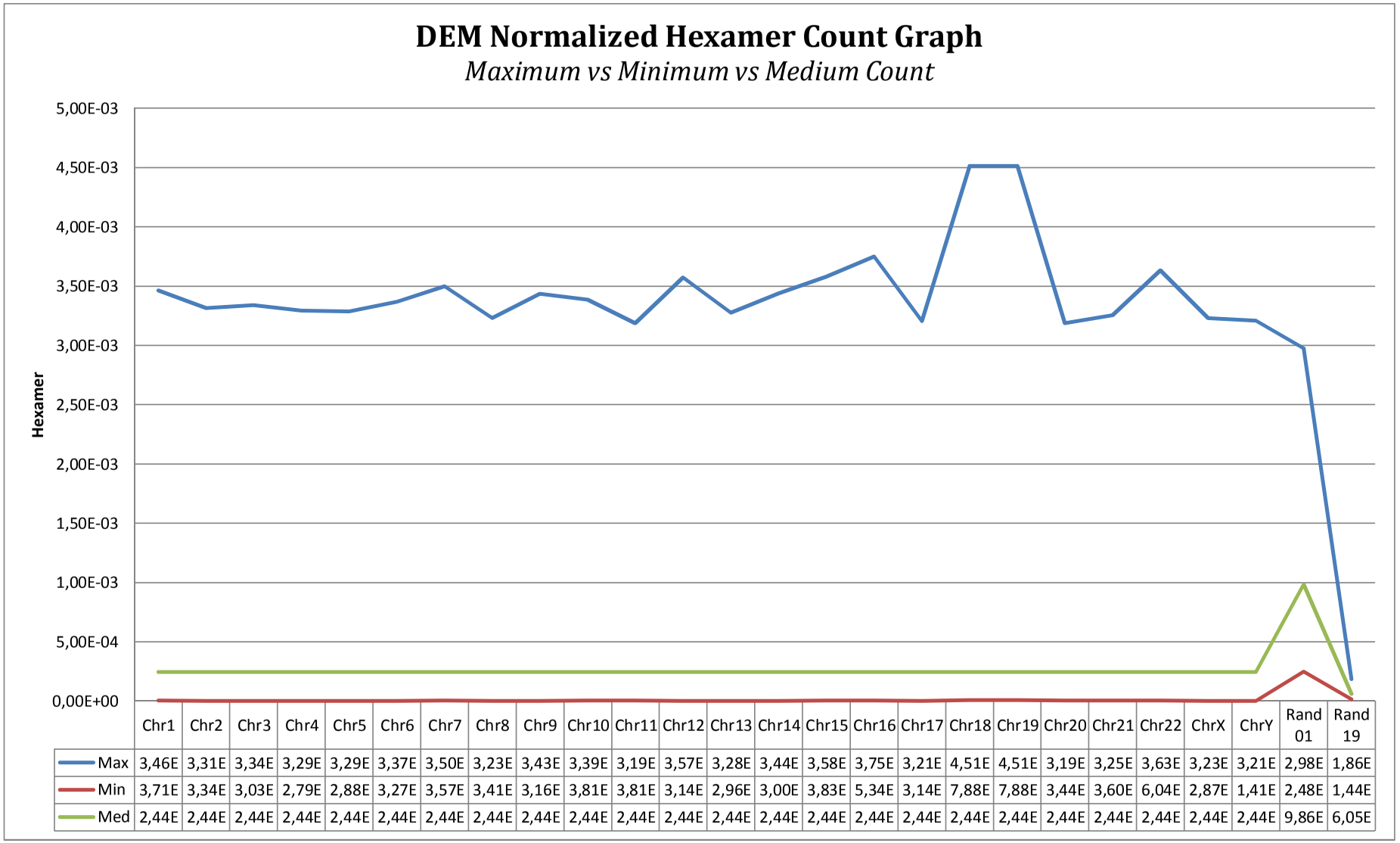
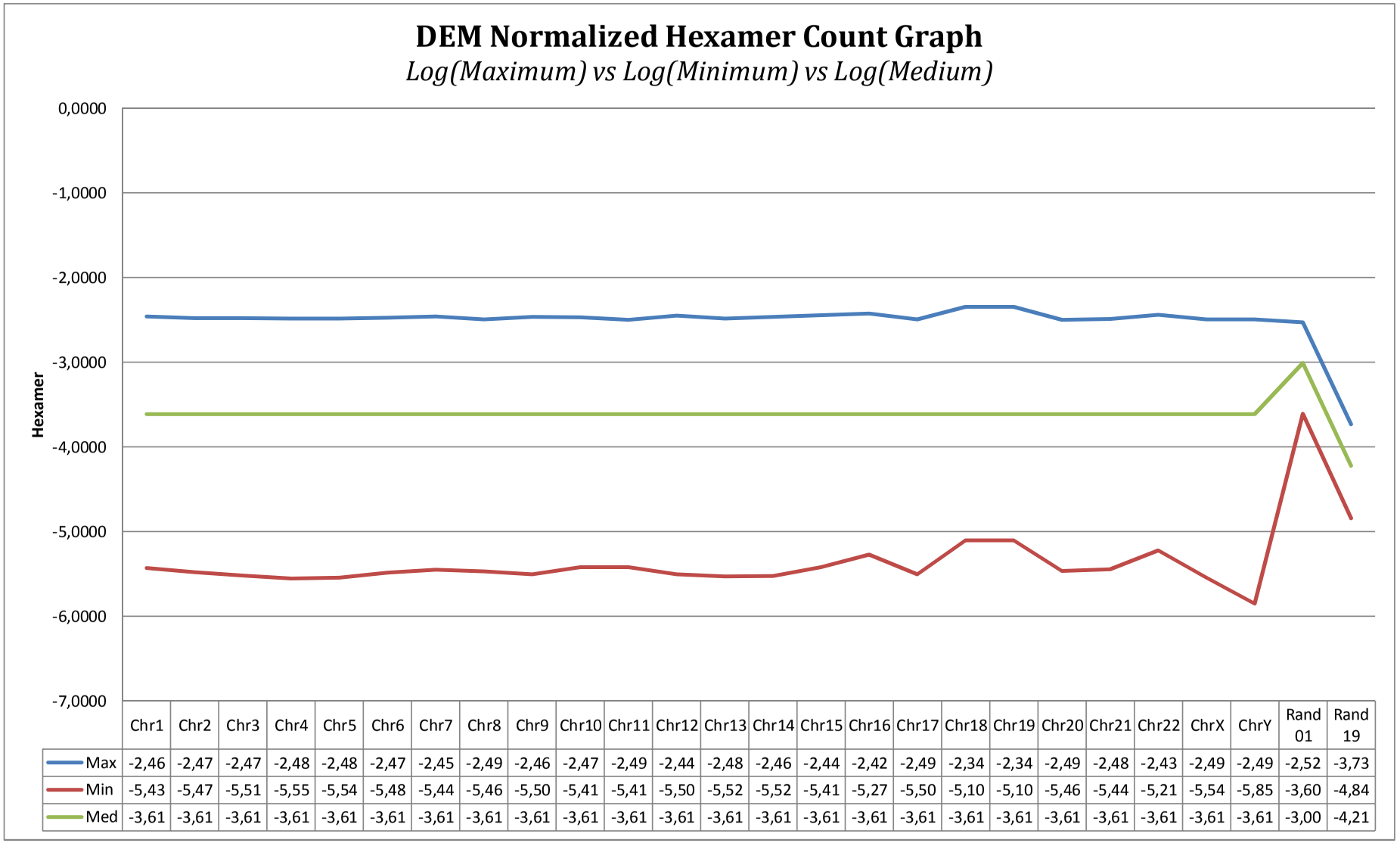


Figure 17. The Zipf elongated hexamers distribution for chromosomes 1 and 19 and for the random sequences Rand1 and Rand19. While the distribution values increase smoothly in the case of the real chromosomes, the random sequences have a stair-like plot.



	Chr1	Chr2	Chr3	Chr4	Chr5	Chr6	Chr7	Chr8	Chr9	Chr10	Chr11	Chr12	Chr13	Chr14	Chr15	Chr16	Chr17	Chr18	Chr19	Chr20	Chr21	Chr22	ChrX	ChrY	Rand 01	Rand 19
Max	3,46E	3,31E	3,34E	3,29E	3,29E	3,37E	3,50E	3,23E	3,43E	3,39E	3,19E	3,57E	3,28E	3,44E	3,58E	3,75E	3,21E	4,51E	4,51E	3,19E	3,25E	3,63E	3,23E	3,21E	2,98E	1,86E
Min	3,71E	3,34E	3,03E	2,79E	2,88E	3,27E	3,57E	3,41E	3,16E	3,81E	3,81E	3,14E	2,96E	3,00E	3,83E	5,34E	3,14E	7,88E	7,88E	3,44E	3,60E	6,04E	2,87E	1,41E	2,48E	1,44E
Med	2,44E	2,44E	2,44E	2,44E	2,44E	2,44E	2,44E	2,44E	2,44E	2,44E	2,44E	2,44E	2,44E	2,44E	2,44E	2,44E	2,44E	2,44E	2,44E	2,44E	2,44E	2,44E	2,44E	2,44E	9,86E	6,05E

Figure 18. The DEM normalized hexamer count graph showing the maximum, average and minimum counts for each chromosome (respectively, in blue, green and red). For real chromosomes all values are approximately constant (with the notable exception of the maximum value for chromosomes 18 and 19); the random sequence Rand1 presents significantly higher minimum and average values and a slightly lower maximum value, while the random sequence Rand19 has much lower values.



	Chr1	Chr2	Chr3	Chr4	Chr5	Chr6	Chr7	Chr8	Chr9	Chr10	Chr11	Chr12	Chr13	Chr14	Chr15	Chr16	Chr17	Chr18	Chr19	Chr20	Chr21	Chr22	ChrX	ChrY	Rand 01	Rand 19
Max	-2,46	-2,47	-2,47	-2,48	-2,48	-2,47	-2,45	-2,49	-2,46	-2,47	-2,49	-2,44	-2,48	-2,46	-2,44	-2,42	-2,49	-2,34	-2,34	-2,49	-2,48	-2,43	-2,49	-2,49	-2,52	-3,73
Min	-5,43	-5,47	-5,51	-5,55	-5,54	-5,48	-5,44	-5,46	-5,50	-5,41	-5,41	-5,50	-5,52	-5,52	-5,41	-5,27	-5,50	-5,10	-5,10	-5,46	-5,44	-5,21	-5,54	-5,85	-3,60	-4,84
Med	-3,61	-3,61	-3,61	-3,61	-3,61	-3,61	-3,61	-3,61	-3,61	-3,61	-3,61	-3,61	-3,61	-3,61	-3,61	-3,61	-3,61	-3,61	-3,61	-3,61	-3,61	-3,61	-3,61	-3,61	-3,00	-4,21

Figure 19. The DEM normalized hexamer count graph where the *y* axis follows a logarithmic scale. The trend for all values is generally constant; we can notice the same increase we observed in the previous graph for chromosomes 18 and 19. Additionally, chromosomes X and Y present a decrease of the minimum count value, and the brisk trend changes for the random sequences is much more noticeable.