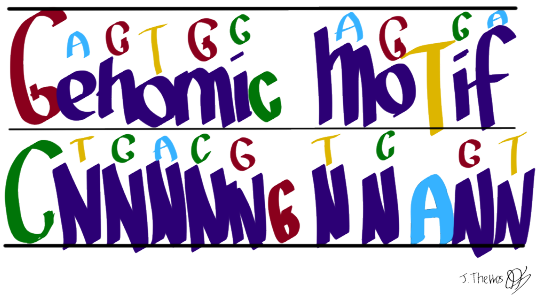
Genomic motifs

Edis Gasanin & Alysée Khan & Mathieu Reibel & Romain Rochepeau & Jade Therras

Team 31



1 Some basics

Living beings are made of one or several cells, all containing the same genetic information. To be able to develop themselves, divide/reproduce or even interact with their environment, complex reactions involving proteins actions occur in each cell.

For those proteins to be created within the cells, some particular DNA sequences (generally between 5-15 BP long) upstream the one that encodes for the wanted molecule, should be bound by specific factors (other proteins). This interaction allows the transcription reaction that will lead to the wanted protein expression.

For searchers, it is interesting to know which DNA sequences are bound by a protein factor to induce a specific gene expression. Through several experiments and sequencing, it has been revealed that the proteins factors do not need to bind a unique and specific DNA sequence to induce gene expression, but allows some base changes into its frame. This discovery leads to the establishment of a consensus sequence.

A consensus sequence representation is made of the bases constituting the DNA binding site sequence. The letter size is proportional to the probability of finding this particular base at this place, knowing a protein factor has bound to it.

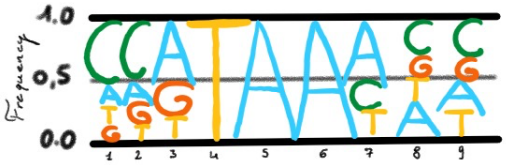


Figure 1: Representation of a consensus sequence: How to interpret it? On position 1 we can have all the bases, but C is the most common base found at this place. On position 3 we cannot find a C base (protein factor would not bound) On Position 4 the only base we can have so the factor binds is a T.

If we have a particular DNA sequence, we could ask ourselves how probable it is that a specific protein factor binds to it. This implies knowing the frequency of each base appearance at each position of the consensus sequence. Through some biological experiments, like SELEX for example, the DNA sequences to which a particular protein factor binds can be found, sequence and map on the genome. Some basic computations can be done to obtain the consensus sequences with each base frequency. Thanks to this first result, we can

compute the "score" of the particular DNA sequence to know if it is probable that the protein factor binds it.

2 Computer Program importance

Even if the statistical computations are not difficult to manage, the study is done on an entire genome. Thus, the amount of data is huge and doing all the computations by ourselves is quite impossible (human genome is around 6.4 billion base-pairs for example). Algorithms help us to do all of this calculations on the large amount of data in a faster and preciser way.

3 Weight-position Matrix to represent the Consensus Sequence

Let's assume we have done experiences and obtained a pool of DNA of different sizes. They all have in common the fact that a particular protein factor has somewhere bound on them. Those sequences are map into the genome, so we have the initial and final positions of the sequence on the chromosome it stands. As already said, the consensus sequence is up to 15 base-pairs but the DNA sequences obtained after the experiments are larger. In addition, we already know that to represent this consensus sequence, we can look at the frequency we encounter a particular base at a given position of the DNA binding sequence.

A way to find it would be to align all the DNA sequences obtained to visualize where there are similar and thus, find the consensus sequence. But another way to do it is to consider the entire sequence we have and do the computations for all sets of following bases of the consensus sequence size we have. Let's go through this algorithm implementation.

4 Class Matrix

This class is made in order to compute the bases frequencies as we discussed before. To create it, we need to have all the DNA sequences that has been somewhere bind within by the protein factor. This amount of data, that helps for the sequence localization in the genome, are collected in a Bed folder.

In other words, each line of the Bed folder contains information about the localization of one particular sequence of DNA that is known to contain the binding site sequence.

```
 chr1   12186   12245  region1  12.5  +
 chr5    4589    4627
chr12  134278  134365  region3   0.0  -
```

Figure 2: Example of Bed folder lines

The first useful information is the chromosome number where the sequence was found. Then, the initial and final positions of the sequence are given. The region, score and sense of the DNA strand can be omitted in the folder. Even tough, the sense of the strand has to be considered in order to be able to do our computations on the good sequence and not an inverted one. A Bedline structure has been created in order to compile all the important piece of information (Chromosome number, initial and final position) of a line together.

The other given folder in the organism genome (Fasta Folder), on which we have done the experiments and obtained the previous data. Recall that the role of the Matrix class is to create the Matrix containing the frequencies we can find a particular DNA base at a given position of the binding site sequence, we can combine the data provide by the Bed and the genome sequence to reach our goal.

Like previously said, the genome of an organism can be huge. Thus, it is not favorable to stock it somewhere in the computer memory. In the other hand, for the program speed, it is important to have the

piece of information we work on at your fingertips. So, in the Make_Matrix method, we use the lire_fasta one (from the Traitement class that we will discuss later) to obtain a line of maximum _MAX_FASTA_ characters from the same chromosome or the chromosome number if the "changement" boolean is true. We will discuss more over this method in the Traitement class paragraph.

To have a fix size of character in our string is important so the programmer know at any time the maximum amount of data he works with and to allow him to have a functional program for every Fasta line size he could encounter.

Another method: read_bed, in the Matrix class, is charged to work on the Bed folder in parallel and to return the positions and sense of a DNA sequence in a Bedline. At a certain time, the actual returned Bedline is always on a further emplacement in the genome than the previous returned one, but before the next one. In other words, the positions and sense of a sequence in the Bed are returned one after the other as the method is called, in an ascending order. This is very important if we do not want to reiterate on our DNA genome. Indeed, if the returned Bedline was not sorted, we could have to search a DNA sequence emplacement that has already been processed. Thus, we would have to reiterate on the Fasta folder to find it, which is bad as we want an efficient (fast) program. So, we have a string that contains a fix number of nucleotides of a particular chromosome we know and the localization of a DNA sequence. Thereby, the global idea is to find if the DNA sequence is in the string and if it does, find it and keep it under hands.

To link the two folders, the find_sequence method is implemented. This one is called when a string has been filled up with nucleotides from the Fasta. In this method, we also call the read_bed one and do comparisons of the chromosome number we have nucleotides from in the string and the one we should take a sequence of with its positions in the chromosome (given by the read_bed method).

If the sequence corresponding to the Bedfile information is found on the string we are working one, the fill_matrix method is called and has to implement the matrix, one nucleotide after the other and according to the sense it should be (inverted or not).

If the matrix is entirely filled, which means that all the Bedfile sequences that stands on the Bedfile has been processed, we can do our final computations on the Matrix to obtain results similar as probabilities (sum of each line equal to 1).

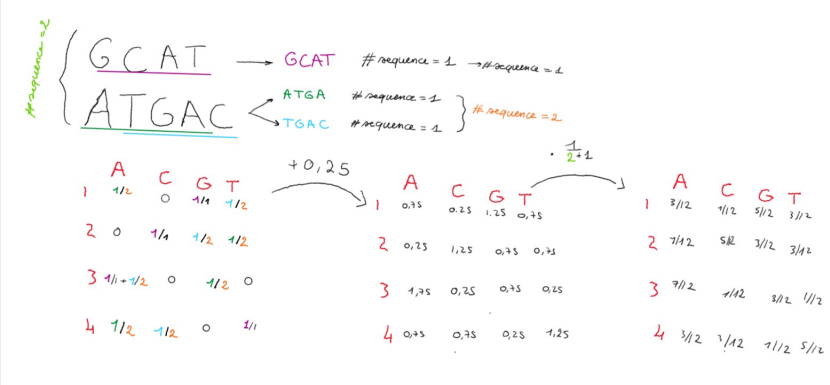


Figure 3: Example of matrix creation having to DNA sequences and a consensus sequence size of 4

Let's briefly explain what are those computations. In fact, it is just adding the value 1 when a base is found, at the proper place (thanks to the conversion method in Traitement) in the matrix. Which is where it was found in the sequence of the consensus size one and in the good column accordingly to the DNA base it is.

If the DNA sequence size is bigger than the consensus sequence size, one should divide the +1 value by the number of different sequences of the consensus sequence size we have within. When all the sequences have been processed, we should add +0.25 to all values and divide them by the number of different sequences we treated (last method). The sum on each line should be one.

5 Class Production

Now, Let's talk about the second implementation possible: if we have the Matrix and the FastaFile and want to obtain the BedFile. In this case we have the probability of a base to bind a special place in the motif sequence and want to find all the sequences in the given genome where the factor have a significant probability to bind. We introduce a treshold in order to assume that if the score of any sequence in the genome is higher than this value, the sequence can be retain as one where the factor could bind. Thus, the aim of the Production class is to return all the sequences with a score higher than this value.

The score is calculated using the calcul_score method and the matrix' values. Thus, Production class has a matrix attribute initialized with the given data. The file returned contains the name of the chromosome, the begin position of the sequence in the fasta file, the sense, the sequence of interest and the score.

In order to iterate on all the genome, the production_file method use the lire_fasta method, and call the traitement_ligne or the changement_chr according to the bool changement.

Traitement_ligne method allows to iterate on a sequence, compute scores and print specific motifs. To prevent the entire genome storage, this method uses a vector of strings. When we encounter a nucleotide, one should add it in each string constituting the vector. When the string reaches the motif size, the score is computed and the write method is called in order to print the motif if needed and finally return it. This implementation allows the file to be written in real time and minimize the storage.

Changement_chr method allows to reinitialize the attributes of the class when we switch chromosome in the fasta file.

6 To link them: Class Traitement

As you may have seen the two classes need to read lines from a fasta folder(lire_fasta). Furthermore, both implementations need a method which returns the position on the matrix for a particular base (conversion method) and to inverse a sequence (inversion method). This are treatments that we do on our fasta, returned sequence or to find Matrix positions. Thus, a class has been created in order to avoid code duplication. This class is the "Traitement" class and contains those methods.

An attribute of the class Matrix is a pointer on a Traitement and enables the use of its method in both Matrix (because it is an attribute) and production (as it has a pointer on a Matrix attribute.).

Let's briefly explain those two methods. The lire_fasta method enables us to obtain a string containing a maximum of _MAX_FASTA_ characters. It uses the getline method with a fixed number of character (initialized at the _MAX_FASTA_ value) and a line break. The lire_fasta method also has a Boolean, which indicates if the line that will be returned by the lire_fasta method is a line with the chromosome title "chrXX". If this Boolean is false, the returned line is a string that contains a maximum of _MAX_FASTA_ nucleotides (it can be less if the chromosome is finished).

The conversion method has a basic implementation with the "switch/case" mechanism. It compares the given nucleotide to one of the cases (it can be A/C/T/G/N) and return the number of the column it is related to.

The inversion method allows to return the complementary sequence of a given sequence.

Finally, in the same way that the two classes need to read a fasta, the two have to write down result. Traitement contain a ofstream attribut, by default std::cout, that can by set by set_outfile.

7 How to run the Program: Classes main and Interface

As the program has two functionalities, the user has to choose one of the two to run the program. To do this, the class Interface is made. This class has to ask the user what functionality he wants to obtain (produce a bedfile or a matrix) and to run the program accordingly to this choice. Thus, the Interface class has a constructor in which a TCLAP has been encoded. This means that the program, as an Interface instance is created, will "ask" by itself to the user what kind of output he wants to have (a bed file or an associated matrix). Furthermore, the user will have to enter where to find the input folders (A Bedfile and a Fasta file to create an associated matrix or a Fasta File and a Bed file to create the associated matrix). Further information could be needed dependently of the output we should obtain.

This piece of information is then set and used to call Interface methods. One of this method is the product_matrix, that take, among others, in parameter the Matrix made with the constructor given the length on TCLAP. Product_matrix has to call on the Matrix the Make_Matrix and the print_matrix methods that will enable to create and fill the matrix, and to print it in a given output file respectively.

As we should create an Interface instance in order to be able to run the program, a main class is done. But more than just being useful to initiate the program, this class enables to manage errors (exceptions) we could have had, trying to set our values using TCLAP.

8 Exceptions and Constant class

Speaking about exceptions, the majority of them have been manage in order to look at the proper opening of our input and output folders. Otherwise, some have been made in order to be sure that no division by zero are made upon the program. We used special errors that have been made on the constant.hpp class and that we can use wherever we want in the program if this folder is included. More than the definitions of the types of exceptions we made to manage the problems we could have while running our program, the Constant.hpp class contains all the needed includes used.

Thus, we just have to include this and not all the others. This will avoid to have circular dependencies.

9 Bedgraph extension

It can be interesting to compare our program results with experimental results. The experimental file results on a "binding score" for all nucleotides in a genome (or a part of a genome), it is the bedgraph file. This extension allows to compute the "area" of a motif probability curve. We take into account 10 nucleotides before and after the sequence, in order to compare our results with the experiment ones.

We essentially encoded it in the Production class, as it is an optional functionality for the output file.

The set_bedgraph method is used when the extension should run. A bedgraph line contains the name of the chromosome, a begin and end position in the genome and the score of a nucleotide (see extension bedgraph instructions). If score is zero, the nucleotide is not shown. Then we use the structure line_bedgraph to represent a line.

It is essential to read the bedgraph file and store line of this file (not all the lines) one at the time. The read_line allows to obtain a new line and store it in act_line attribute.

We use a vector of data, in which one case represents one nucleotide. When we add a new nucleotide, the edit_table method iterate on the vector to add the new value and pop_up the obsolete one. If needed, we read a new line from the bedgraph. When we have to write a line, the surface method just have to sum the values that stand in the vector.

10 EM_algorithm

The goal of this algorithm encoded in the Interface class is to produce a matrix and then iterate on the initial sequences of interest to find the motif with the best score in each sequence. Then a matrix is made with those motifs and we try back to find the motifs with best score according to this new matrix. We do this until the matrix values do not change anymore or that the algorithm has run the maximum of times we decided.

By default the fasta ouput name is interest sequences.fasta.

The algorithm start with the production of a matrix using the product$_matrixmethod.Productionusethismatrixandthefast$

11 Tests and run the program

All over the program conception, tests should be made in the test folder. There are written in the test main.

In this folder, we have created some instances: Traitement, Production, line_bedgraph, Matrix and some strings. All of those instances have known values in them, which will allow us to verify if our methods give the results we want. In addition, a structure of four Bedline is created in order to test the read_bed and compare_lower/_higher methods.

We will not go through all the tests that have been implemented, as all the methods that do not have a trivial algorithm should be tested. We don't test the method where results can be directly seen in the terminal (for example, method product_file() or Make_Matrix() cause it's only compiling other methods, or print methods)

All those tests stand on the comparison between the result of a method and the known result it should give. For example, the conversion method test of the Traitement class compare the result of the conversion() returned value of a known string with a double value. This comparison is done with the EXPECT_EQ as we test a method that return a value and the double value to which we compare it is the expected output of the conversion method. For the lire_fasta method test, we test if the Boolean that change is the method have the expected value given a fasta folder we made. We also compare what contains the string at a particular time, testing if the lecture line is really equal to the expected one.

To run the program, one should give the necessary information when entering the command:

./Genomic_Motifs -h from the build folder

Depending of the output that should be obtained, different parameters should be set. For example, to run the program with the given test files:

To obtain a BedFile:

../Genomic_Motifs -F -f relative_path/promoters.fasta -m relative_path/DBP.mat -T 0
(-o relative_path_where_want_folder_to_be output.txt -B relative_path/bedgraph file)

ex of bedgraph for this fasta : ../test/test_bedgraph.bedgraph

To obtain a Matrix:

../Genomic_Motifs -M -l 6 -b relative_path /BMAL1_chr7.bed
-f relative_path /chr7.fa (-o relative_path_where_want_folder_to_be/output.txt -E for EM_algo)

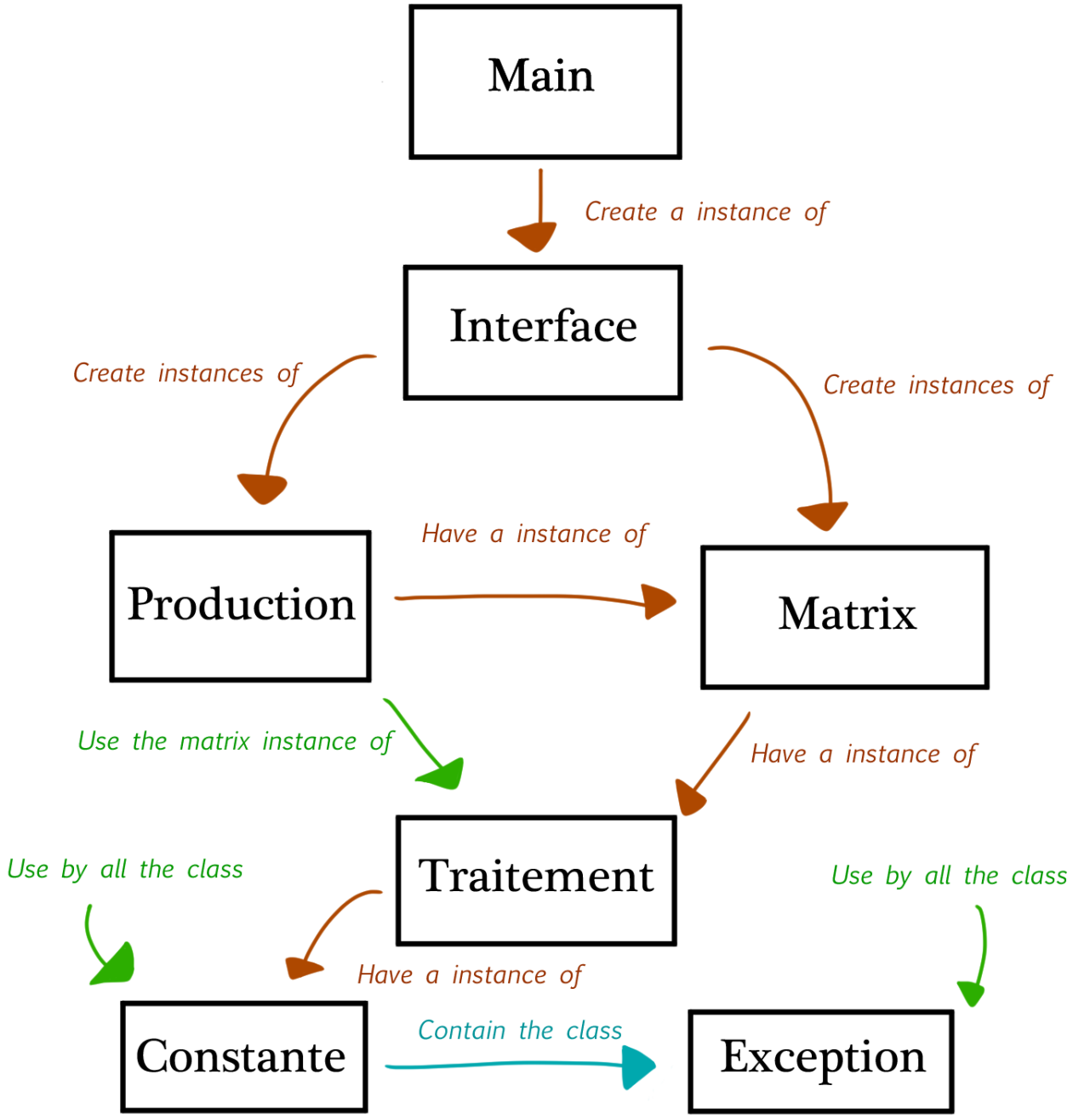


Figure 4: Hierarchic diagram