1 Data representation

1.1 sequence file

A sequence file is a text file of l lines, each line have only one integer, representing the character(chromatin state) of this site. Usually the file name ends with ".seq".

1.2 compressed sequence file

A compressed sequence file is a compressed representation of a sequence file. It is a k line text file, in which each line has two integers: "(character) (repeated time)".

Usually the file name ends with ".sseq".

1.3 sequence database

A sequence database is composed of n compressed sequence files and one index file. The compressed sequence files should locate in the same folder.

The index file stores the paths to the compressed sequence files, each file name takes one line.

A sequence database could be a set of genome regions, an epigenome, a mixture of several epigenomes, or whatever you can come up with.

Usually an index file has no suffix.

2 Alignment algorithms

2.1 Smith-Waterman algorithm for chromatin state sequence alignment

2.1.1 Definition of alignment problem

In this subsection, a chromatin state sequence is represented by a compressed format:

$$Seq := \{n, S, L\}$$

n is the length of array S and L, S is an array of chromatin states, and L an integer array. Inside L, L_i is the repeat time of chromatin state S_i. For example: $\{3, [a, b, c], [1, 2, 3]\} = [abbccc]$.

The i'th element of a sequence Seq, $[S_i, L_i]$ is a chromatin segment. Our goal here is to align the segments between two sequences.

Given two chromatin state sequences $Seq_1 = \{n_1, S^{(1)}, L^{(1)}\}$, $Seq_2 = \{n_2, S^{(2)}, L^{(2)}\}$, a match f is a string. On each position of f, f_i has three choices: 'm', 'g_1', or 'g_2'. 'm' is a match, 'g_1' is a deletion in Seq_1 and 'g_2' is a deletion in Seq_2. The number of 'm' plus the number of 'g_1' should equal to n_1. Symmetrically the number of 'm' plus the number of 'g_2' should equal to n_2.

Let u^1 and u^2 be the index of f on Seq_1 and Seq_2. u^1_i could either be an integer or $\varnothing$, indicate which segment of $Seq1$ f_i is pointing to, or f_i is pointing to a gap.

Now we are able to define the matching score H given Seq_1, Seq_2 and f.

$$H(Seq_1, Seq_2, f) = \sum h(f_i, u_i^1, u_i^2, Seq_1, Seq_2)$$

If f_i equals to 'm', $h(f_i, u_i^1, u_i^2, Seq_1, Seq_2) = MF(u_i^1, u_i^2, Seq_1, Seq_2)$, MF is the matching function, otherwise, f_i equals to 'g_s', then $h(f_i, u_i^1, u_i^2, Seq_1, Seq_2) = GF(u_i^s, Seq_s)$, GF is the gap function.

In alignment problem, we want to find the best match f^*, which maximize the alignment score.

Alignment is achieved by dynamic programming. This algorithm iteratively maintain and update a matrix M:

$$\begin{aligned}
&M(i,0) = 0\,, \text{ for } 0 \le i \le n_1\,;\\
&M(0,j) = 0\,, \text{ for } 0 \le j \le n_2\,;\\
&M(i,j) = \max \begin{cases} M(i-1,j-1) + MF\,(i,j,Seq_1,Seq_2) & \text{(Mis)match}\\ M(i-1,j) + GF(i,Seq_1) & \text{Deletion in } Seq_1\\ M(i,j-1) + GF(j,Seq_2) & \text{Deletion in } Seq_2 \end{cases},\\
&\qquad \text{for } 1 \le i \le n_1,\ 1 \le j \le n_2\,,
\end{aligned} \tag{1}$$

M_{ij} is the maximal alignment score of the two subsequences $Seq_1^{[1,i]}$ and $Seq_2^{[1,j]}$.

2.1.2 Local alignment

In Practice, we prefer local alignment rather than global alignment described above. To achieve the goal of local alignment, we need to add a small modification to the dynamic programming.

$$\begin{aligned}
&M(i,0) = 0\,, \text{ for } 0 \le i \le n_1\,;\\
&M(0,j) = 0\,, \text{ for } 0 \le j \le n_2\,;\\
&M(i,j) = \max \begin{cases} 0 & \text{Restart the alignment from here}\\ M(i-1,j-1) + MF\,(i,j,Seq_1,Seq_2) & \text{(Mis)match}\\ M(i-1,j) + GF(i,Seq_1) & \text{Deletion in } Seq_1\\ M(i,j-1) + GF(j,Seq_2) & \text{Deletion in } Seq_2 \end{cases},\\
&\qquad \text{for } 1 \le i \le n_1,\ 1 \le j \le n_2\,,
\end{aligned} \tag{2}$$

2.2 Attention score

Before defining the Matching function and Gap function, we want to define the attention score first. Based on an observation: given a sequence Seq, some segments in Seq could be more important than other segments, we want to score the importance of each segment in Seq. The attention score is an array of float, denoted by $AS(Seq)$. Furthermore, the i'th element of $AS(Seq)$ is denoted by $AS(Seq, i)$.

In our package, we provided three kind of attention scores:

Naive attention score:

$$AS^{n}(Seq, i) = 1$$

Minus log frequency(of chromatin states):

$$AS^{lf}(Seq, i) = AS^{lf}(\{n, S, L\}, i) = -log(frequency(S_i))$$

Frequency of the chromatin states are computed outside and passed to the alignment functions.

Unpredictability:

$$AS^{up,w}(Seq, i) = GramScore^{w}(Seq, i) * LocalRankScore^{w}(Seq, i) * LengthScore(Seq, i)$$

In this definition, we aim to quantify how much the neighbourhood area $Seq^{[i-w,i+w]}$ predicts S_i. The unpredictable segments are thought to be more important than predictable segments.

In the subsequence $Seq^{[i-w,i+w]}$, if neither of pattern $S_{i-1}S_i$ nor S_iS_{i+1} occurs elsewhere, $GramScore^{w}(Seq, i) = 1$, if both patterns occurs elsewhere, $GramScore^{w}(Seq, i) = 4$, otherwise $GramScore^{w}(Seq, i) = 2$.

For $Seq^{[i-w,i+w]}$, we sort the chromatin states occurred in this region by their frequency, $LocalRankScore^{w}(Seq, i) = 1 + \frac{rank(S_i)-1}{|\{States \in Seq^{[i-w,i+w]}\}|-1}$

If $L_i = 1$, $LengthScore(Seq, i) = 0.5$, otherwise $LengthScore(Seq, i) = 1$.

2.3 Matching function

$$MF(Seq1, Seq2, i, j) = \begin{cases} AS(Seq1, i) + AS(Seq2, j) & S1_i = S2_j \\ -\epsilon * (AS(Seq1, i) + AS(Seq2, j)) & S1_i \not= S2_j \end{cases}$$

2.4 Gap function

$$GF(Seq, i) = -\epsilon * AS(Seq, i)$$

2.5 Bag of Words alignment

3 C codes

Matching score and attention score are defined in "/Ccode/custom/CustomFunction.h(.c)". A string "para" is passed into these functions. "para" could be a value, a path to a file, or anything.

Here we provide 3 types of attention scores in our package, as what has been described in the previous section.

Attention Score 1: constant. Path: "/Ccode/custom/CustomFunction.Naive.c"

Attention Score 2: log frequency. Path: "/Ccode/custom/CustomFunction.Frequency.c"

Attention score 2: unpredictability. Path: "/Ccode/custom/CustomFunction.Attention.c"

You can also write your own CustomFunction.c. Make sure "/Ccode/custom/CustomFunction.c" link to your code you want to specify before compile.

3.1 folder structure

There are 6 useful folders here. They are "/Ccode/bin", "/Ccode/main", "/Ccode/include", "/Ccode/custom", "/Ccode/Alignment", "/Ccode/DataProcessing".

"/Ccode/bin" contains executable files and bash scripts. "/Ccode/main" contains interface codes. The other folders have some basic library functions.

3.2 compile

The script "compile.bash" compiles the C codes into executable files to folder "/Ccode/bin". Before running this script, all codes in the other folders should have a link in folder "/Ccode/main".

"FakeChromosomeGenerator.cpp" uses random generator specified by c++11, hence the compiler should support c++11. In linux, g++ version 4.8 or higher supports c++11, remember to substitute "g++-5" in "compile.bash" to your compiler.

3.3 bash scripts and executable files

3.3.1 TowRegions.out

Usage: ./TwoRegions.out seqfile1 seqfile2 para

This program implement smith-waterman algorithm to compare two epigenetic state sequences. "seqfile1" and "seqfile2" are two compressed sequence files. "para" is defined at the beginning of section 3.

The output includes the best local match regions and the matching score.

3.3.2 DatabaseSearch.out

Usage: ./DatabaseSearch.out Paths_Search Para_Search Para_align

This program is doing search of a query sequence set from a database. Both query sequence set and the database are stored as "sequence database". There are examples of "Paths_Search", "Para_Search" under "/Ccode/bin". "Para_align" is the same as the "para" described at the beginning of section 3.

3.3.3 DatabaseSearch_baseline.out

Usage: ./DatabaseSearch_baseline.out Path_Search_baseline Para_Search_baseline

3.3.4 FakeChromosomeGenerator.out

Usage: ./FakeChromosomeGenerator.out chromosome fake_chromosome

This program generate fake chromosome by Markov rule. Fake chromosome has the same length to the original. The state transition probability matrix to generate fake chromosome is computed from the original chromosome.

"chromosome" is a compressed sequence file(of epigenetic states), "fake_chromosome" is the file to output.

3.3.5 cut_sseq.out

Usage: ./cut_sseq.out Para_Cut
Cut a long sequence into regions, each region stored in a file.

3.3.6 CutFolder_Init.sh

3.3.7 FakeGenomeGenerator.sh

Usage: ./FakeGenomeGenerator.sh index genomefolder newindex newgenomefolder
A genome has more than one chromosomes, this script iteratively run FakeChromosomeGenerator.out on the chromosomes. The genome is a sequence database structure.

3.3.8 GenomeSearch_Path.sh

Usage: ./GenomeSearch_Path.sh Para_GenomeSearch PathToThisScript
Automatically cut query genome into regions, and do search for each query region from another genome.

3.3.9 Algn2AnoBatch.sh

3.4 library functions

3.4.1 StateIO.h

The original code of this library is under "/Ccode/DataProcessing". The functions inside this library do file reading tasks.
Sseq_ReadFile: ArrayLength = Sseq_ReadFile(FileName, StateArray, LengthOfEachState, opt) .
Read compressed sequence file and record the state sequence into arrays.
LengthOfEachState is an array of unsigned short, state segments longer than "USHRT_MAX" are cut into smaller segments.
Seq_ReadFile: ArrayLength = Seq_ReadFile(FileName, StateArry, opt) .
Read sequence file and record the state sequence into an array.
Lines_ReadFile: LineNumber = Lines_ReadFile(FileName, Lines, opt=NULL) .
Read a file line by line and record the lines as strings, '\n' are removed.

3.4.2 WatermanFun.h

The original code of this library is under "/Ccode/Alignment". The functions here do ordinary alignment, bag of words alignment, besides, structures and functions assist alignment are also included here.
SWA_Even:
Trace_Even:
Print_Alignment_Even:
Print_Alignment_Sseq_Even:
SWA_Bow_Even:
Trace_Bow_Even:

3.4.3 CustomFunction.h

3.5 Example

Here we illustrate how to use our programs by running some examples.

To be convenient, the epigenome here only have 2 chromosomes.

At first we need to compile the codes. In this section naive attention score is used, link "/Ccode/custom/CustomFunction.c" to "/Ccode/custom/CustomFunction.Naive.c".

Recommended command: ln -sf /Ccode/custom/CustomFunction.Naive.c /Ccode/custom/CustomFunction.c

Then run "./compile.bash" under "/Ccode/main". Make sure your g++ supports c++11.

3.5.1 Align two epigenetic state sequences

Command: /Ccode/bin/TwoRegions.out /Ccode/example/seq1 /Ccode/example/seq2 1.0

1.0 is the ϵ used by matching function and gap function described in section "Alignment algorithms".

3.5.2 Generate fake epigenome

Enter directory /Ccode/example .

Command: /Ccode/bin/FakeGenomeGenerator.sh epigenomefiles epigenome/ fakeepigenomefiles fakeepigenome/

The script FakeGenomeGenerator.sh will try to remove folder "fakeepigenome" first, it may give you warning if this folder doesn't exist at all.

3.5.3 Cut an epigenome in to small regions

Enter directory /Ccode/example .

Command:

/Ccode/bin/CutFolder_Init.sh Para_Cut.example

/Ccode/bin/cut_sseq.out Para_Cut.example

"Para_Cut.example" is a text file. There's annotation inside "/Ccode/example" . The small regions are selected as the following:

$$chr_i^{[0,window)}, chr_i^{[step,step+window)}, chr_i^{[2*step,2*step+window)}, ...$$

3.5.4 Query database search (with baseline database)

As mentioned in section "Ccode, DatabaseSearch.out", both query and database are stored as "sequence database" defined in section "Data representation, sequence database". Our code iteratively select the sequences in the first database, for each selected sequence, we implement our alignment algorithm on the second database and record the top k hits to the output file corresponding to this selected sequence.

We want to restrict the top hits such that they are apart from each other. Let the length of a query sequence to be l. Indexed in the database, a hit starts from s_h, ends at t_h, then from $s_h - a * l$ to $s_h + b * l$ there should be no other hit.

We also need you to provide a baseline database, each sequence in query database are also aligned to the baseline database. However, for whatever baseline database you provide, it doesn't alter the alignment result.

Command:

mkdir epigenomechr1align

/Ccode/bin/DatabaseSearch.out Paths_Search.example Para_Search.example 1.0

1.0 is the ϵ used by matching function and gap function described in section "Alignment algorithms". You can find annotations of "Paths_Search.example" and "Para_Search.example" in "/Ccode/example/"

The summary is shown at "stdout". If there were n sequences in the query database, the summary would be an $n*(k+2)$ matrix. Form column 1 to column end are: length of this query sequence(compressed form), best alignment score to baseline set, top k alignment scores to the dataset.

For each query sequence, an output file is created to record its top hits. Explanation of these files is in "/Ccode/example/Record_Alignments.annotation"

3.5.5 Query database search (without baseline database)

Basically this is identical to the previous example, but baseline database is no longer needed here. The top k alignments (for technical reasons, the program output more than k alignments, only the top k are correct) for each query sequence are same to the alignments found by the previous example.

Command:

mkdir epigenomechr1alignnative

/Ccode/bin/DatabaseSearchNative.out Paths_SearchNative.example Para_SearchNative.example 1.0

3.5.6 Horizontal alignment across an epigenome (with baseline)

In this example, we cut the epigenome into small regions first, for each small region, the top hits in this epigenome are searched afterwards.

This goal could be achieved by running the previous example chromosome after chromosome. Here we provide an integrated script.

Command:

mkdir horizontalalignment

/Ccode/bin/GenomeSearch_Path.sh Para_GenomeSearch.example /Ccode/bin/

"/Ccode/bin/" is the path to the bash script.

3.5.7 Horizontal alignment across an epigenome (without baseline)

This example is almost identical to the previous example, but baseline database is no longer needed here. The top k alignments for each segments are same to the alignments found by the previous example.

Command:

mkdir horizontalalignmentnative

/Ccode/bin/GenomeSearchNative_Path.sh Para_GenomeSearchNative.example /Ccode/bin/

"/Ccode/bin/" is the path to the bash script.

3.5.8 Annotate the outputs of horizontal alignment

In this example, we use "Algn2AnoBatch.sh" to annotate the outputs of horizontal alignments. For each output file of horizontal alignment, "Algn2AnoBatch.sh" does 3 things: sort the hits by alignment score, coordinating the hits to hg19, and mapping the chromosome index to chromosome name.

After the annotation, copy the given positions of query sequences and hits into UCSC website for detailed information:

http://genome.ucsc.edu/cgi-bin/hgGateway?redirect=manual&source=genome.ucsc.edu

Command:

/Ccode/bin/Algn2AnoBatch.sh Para_GenomeSearch.example Para_Annotation.example

or

/Ccode/bin/Algn2AnoNativeBatch.sh Para_GenomeSearchNative.example Para_Annotation.example

For each output file of horizontal alignment, the annotated file is under the same folder with the suffix ".anno".

4 matlab code

5 julia code

6 python code