Applied Bioinformatics

of Nucleic Acid Sequences

David A. Hendrix

February 14, 2018

Contents

Chapter 1

Introduction to Biological Sequences

1.1 Nucleic Acid Bioinformatics

The tremendous growth of bioinformatics and computational biology in the late 20th and early 21st centuries has had an associated growth in software and algorithms for studying biological sequences. These class notes are designed to introduce students of the life sciences with little to no programming experience to the concepts and methodologies of bioinformatics, and contemporary software applications. The majority of this course will deal with the analysis of nucleic acid sequence data. Many of these applications have web interfaces, but where possible command line software and strategies for dealing with sequence data with the GNU/Linux command line interface (the terminal) will be used. We won't be content to simply run commands blindly, but rather we will also learn a great deal of the equations and theory behind these methods. In addition, these notes will provide an introduction to Biopython as a tool for bioinformatics and as a framework to connect all the concepts presented.

1.1.1 GNU/Linux and the command line

In order to do bioinformatics on more than just sequences that we type ourselves, we need to know how to read and write sequence file formats. And before we can work with file formats, we'll need a brief introduction to the GNU/Linux command line interface, which is very well suited for working with these kinds of files. In what follows is a very basic introduction. We will continue to learn new GNU/Linux commands while we broaden our repertoire of software tools, Biopython commands, file formats, and databases.

The GNU/Linux command line interface is a text-based interface to files and commands. Such a terminal can be accessed from many terminal applications in a GNU/Linux distribution, or with the Terminal program (and others such as iTerminal) for Mac OS X, and from many emulators available for Windows, such as Cygwin. Advanced users of the terminal can accomplish difficult tasks with amazing efficiency, such as running complex commands on thousands of files in one fell-swoop while barely breaking a sweat. The combination of a strong knowledge of the terminal and shell scripting, along with knowledge of a scripting language like python or perl can be very powerful and worth learning for any student of bioinformatics. Scripting languages are named after the fact that they are used to write "scripts", which are programs written to automate a sequence of commands and typically use an interpreter rather than a compiler.

Once you've opened your terminal application, you should be given a cursor in which to type text-based commands. For example, say you want to print a list of of all files and directories in your current location (directory), you use the command `ls`. Directories are like folders for keeping files, but they can also be a location. In other words, you can be working in a directory, and any files you create will then be kept in that directory. A file, such as a text file for storing a sequence, is the unit in which data is stored. Although most people know what a file is from their experiences with computers, there is a lot that could be said about what a file actually is beyond the scope of these class notes. For our purposes right now, this intuitive concept of a file that most people have will suffice.

We can create a directory with the command `mkdir`. For example, let's begin by creating a directory called "scripts" and another called "data", followed by the command `ls` to see the contents of the current

directory:

```
$ mkdir scripts
$ mkdir data
$ ls
data scripts
```

The dollar sign at the beginning of each line is not to be typed, but rather signifies the line where commands are typed, hence the name "command line". The current directory, or "working directory" is the directory that we are in, and upon which all commands we run will act on. We can then see what directory we are currently in with the command `pwd`, which is short for "print working directory":

```
$ pwd
/home/dhendrix
```

We can then change directories with the command *cd*, which is short for "change directory". For example, let's use this command to go into the scripts directory:

```
$ cd scripts
$ pwd
/home/dhendrix/scripts
```

So with these examples, it is clear that directories can contain files and other directories, but they also represent an abstract location in which you are working.

We can create a file in countless ways, Let's start with the creation of a basic text file. There are many programs called text editors that can be used to create a text file, such as `nano`, `emacs` and `vim`. For `emacs`, we can create a file by simply typing `emacs filename`, where `filename` is the file we are creating. Since we are in the scripts directory, let's create a simple python script called "helloworld.py", which as the name suggests is a "hello world script". This is the first script that people typically learn in a programming language because it is the simplest script that actually does something. To do this, while still in the scripts directory, let's type `emacs helloworld.py`, and type the following code:

```
print "hello world"
```

Then to save, we can type `Ctrl-X Ctrl-S` (or the Control key and X at the same time, followed by the Control key and the S at the same time). After it is saved, we can type `Ctrl-X Ctrl-C` to close the program. We can then run our new script by typing the command `python helloworld.py`, which should produce the following output:

```
$ python helloworld.py
hello world
```

If you've never programmed before, you've just created your first script! The preceding was by no means a complete introduction to GNU/Linux. If anything, you should now have a rudimentary starting point. Throughout these class notes, we will learn new commands as we go, and new python code as we go. The intent here is to learn the essential pieces to do something simple, and keep learning as we go. In the end, we'll have a decent set of commands to accomplish a lot.

1.2 Sequences, Strings, and the Genetic Code

1.2.1 Introduction to Sequences and Biopython

Central to the subject of bioinformatics is the study of sequences, and sequence analysis. So we begin with the notion of the biological sequence. Consider a character string x such as

$$x = \texttt{ATGGCGGGAAAATGA}$$

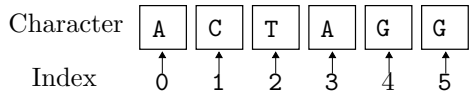

Figure 1.1: A simple representation of a character string,

Obviously, this sequence of characters is very different from a biological polymer such as DNA or protein, but this sequence is useful for the purposes of studying the bioinformatic properties of the molecule it represents. In the language of computer science, these character sequences are often called "strings". We can represent a particular position i of the sequence as $x[i]$. For example, in this example, $x[5]$ is `C`. That is, when using a 1-based position, where the first position is numbered 1, the second position is 2, and so on. Although this may seem intuitive, most programming languages use 0 as the start position.

Let's consider how sequences are represented in python. First, from a terminal, we can enter a python terminal by simply typing "`python`" on the command line. This python terminal is like a sandbox in which we can create sequences and perform different sequence analyses. The same code we type in this python terminal should work in a python script, if we just save the same series of commands in a text file. We can define a sequence and access a position as follows:

```
>>> x = "ATGGCGGGAAAATGA"
>>> x
'ATGGCGGGAAAATGA'
>>> x[5]
'G'
```

Therefore, in python strings are 0-based and begin with position 0, then the second position is 1, the third is 2 and so on. Defining the sequence in Biopython isn't that much different:

```
>>> from Bio.Seq import Seq
>>> x = Seq("ATGGCGGGAAAATGA")
>>> x
Seq('ATGGCGGGAAAATGA', Alphabet())
>>> x[5]
'G'
```

Perhaps the difference is that Biopython takes the concept of a sequence, and makes it an "object". Object oriented programming is a paradigm in computing where the central concept is the "object", and there are defined "methods" that can be applied to these objects. Objects are defined as you might expect, as discrete units that fit some set of parameters or specifications as defined by a "class". For example, the sequence object `x` defined above has some similar features that a physical DNA molecule might have. We can transcribe our DNA sequence `x` by simply typing:

```
>>> x.transcribe()
Seq('AUGGCGGGAAAAUGA', RNAAlphabet())
```

In this example, `x` is the object, and `transcribe()` is the method that is called on the object. Much of python is object oriented, and Biopython is very object oriented. We can see that this particular object is called a `Seq` object. Note that in order to use Biopython we need to import it. We generally need to import a library or module with such a statement in order to use the object defined by them.

The advantage of importing libraries like this is that we get to use built in functions, methods, and objects. For example, a common quantity that one might be interested in regarding a nucleic acid sequence is its GC content. The GC content of a sequence is the percentage of the nucleotides in the sequence that are either G or C. Probably the easiest way to compute it is to import a function from another Biopython module:

```
>>> from Bio.SeqUtils import GC
```

```
>>> from Bio.Seq import Seq
>>> x = Seq("ATGGCGGGAAAATGA")
>>> GC(x)
46.666666666666664
```

1.2.2 The Central Dogma

Let's now consider the central dogma of molecular biology. The central dogma states that biological information generally flows form DNA to RNA to proteins [1,2]. Similarly, with Biopython we can create objects corresponding to DNA, RNA, and Protein sequences. We can use methods on these objects to transcribe the DNA, and to translate the RNA to a protein using the `transcribe` and `translate` functions, respectively. Here's how it's done:

```
>>> from Bio.Seq import Seq
>>> DNA = Seq("ATGGCGGGAAAATGA")
>>> DNA
Seq('ATGGCGGGAAAATGA', Alphabet())
>>> mRNA = DNA.transcribe()
>>> mRNA
Seq('AUGGCGGGAAAAUGA', RNAAlphabet())
>>> protein = mRNA.translate()
>>> protein
Seq('MAGK*', HasStopCodon(ExtendedIUPACProtein(), '*'))
```

The python examples presented so far have been typed into the python terminal, but all of it could have been typed into a text file using one of many available text editors mentioned above, and run on the command line. For example, if the above code (all the stuff after the "`>>>`" was typed into a text file called "`translate.py`", it could be run with the command "`python translate.py`". As it stands right now, it wouldn't print anything, highlighting one distinction between the python terminal and a python script; the variable name alone will not print its contents to the screen, but the print function will. Therefore the following code

```
from Bio.Seq import Seq
DNA = Seq("ATGGCGGGAAAATGA")
print DNA
mRNA = DNA.transcribe()
print mRNA
protein = mRNA.translate()
print protein
```

will produce the following output

```
ATGGCGGGAAAATGA
AUGGCGGGAAAAUGA
MAGK*
```

An additional difference is we are not given the data type information that we get when using the python terminal. Printing the `Seq` object just gives the actual string of that sequence.

Exercise 1

Transcribe the following DNA sequence and translate the resulting mRNA sequence. What protein sequence do you get?

```
>CDS
ATGCTGCGGCGAGCTCCGTCCAAAAGAAAATGGGGTTTGGTGTAAATCTGGGGGTGTAATGTTATCATAT
AAATAAAAGAAAATGTAAAAAACAAAAACAAAAACAAAAAAGCC
```

1.2.3 Subsequences and Reverse Complement

Subsequences for python strings work the same way as subsequences in `Seq` objects. To specify a substring, we need to give a start and a stop position. The resulting substring will consist of the characters from the start to, but not including the stop. That is, the stop position is "non-inclusive". By "non-inclusive", it means it will contain the characters up to, but not including the stop position. This may seem strange at first, but there are some reasons for this that will become evident soon. For now, let's just take a look at an example:

```
>>> from Bio.Seq import Seq
>>> DNA = Seq("ATGGCGGGAAAATGA")
>>> D1 = DNA[4:7]
>>> D1
Seq('CGG', Alphabet())
```

So, by specifying the range `4:7`, we are including positions 4, 5, and 6. The resulting substring is essentially a concatenation of the characters at these positions. What if we create a substring that is the whole string itself? The result is like this:

```
>>> x = Seq('ACGTACGTACGT')
>>> y = x[0:len(x)]
>>> y
Seq('ACGTACGTACGT', Alphabet())
```

There are a couple of things going on here. First we create the string `x` that is of length 12. We then create a substring `y` that starts at position 0 and goes up to, but not including, the length of `x`, which is specified by `len(x)`. Since the indices of `x` go from 0 to `len(x)-1`, the substring `y` is the same as `x`.

If we want to print ever other character, we add to our range the step size, in this case 2.

```
>>> from Bio.Seq import Seq
>>> x = Seq('ACGTACGTACGT')
>>> y = x[0:len(x):2]
>>> y
Seq('AGAGAG', Alphabet())
```

It's important to note that we don't need to enter the start and stop position when we are traversing the whole sequence from `0` to `len(x)`. We can accomplish the same thing with `x[::2]` because the default option is to traverse the whole sequence, so with the start and stop missing it is implied that it is from `0` to `len(x)`.

```
>>> x = Seq('ACGTACGTACGT')
>>> y = x[::2]
>>> y
Seq('AGAGAG', Alphabet())
```

Finally, we can reverse a sequence by using a negative step like -1. By using the default options on start and stop, we can get the whole sequence in reverse

```
>>> x = Seq('ACGTACGTACGT')
>>> x[::-1]
Seq('TGCATGCATGCA', Alphabet())
```

When using starts and stops in reverse, keep in mind that the same policy of going to, but not including, the stop position is used.

Because DNA is typically double stranded, we can think of the DNA sequences as being double stranded with the second strand implied. That is, even though we are given one strand of DNA, it is typically implied that there is an associated second strand that is the reverse complement of the first. We typically use the

`Seq` object to represent the forward strand of our sequence, but we may also want to know what the reverse strand would look like. We can produce the reverse complement very easily with the built-in Biopython method "`reverse_complement()`"

```
>>> x = Seq('ACGTACGTACGT')
>>> x.reverse_complement()
Seq('ACGTACGTACGT', Alphabet())
```

In this case, we see that the reverse complement of our sequence x is the same as x. Such DNA sequences that are equal to their reverse complements are called "palindromes". Regular palindrome phrases such as "race car" simply use the reverse, but DNA palindromes use the reverse complement.

Exercise 2

You can concatenate two sequences (attach them to each other to form one longer sequence) with the "`+`" operator, such as `D = D1 + D2`. How can you use this to create a very long palindromic sequence?

1.3 Sequences File Formats

1.3.1 Fasta

Probably the most commonly used file format for sequences, and in fact one of the most common file formats of any kind in bioinformatics, is the FASTA file format. The FASTA file format has its origins in the program FAST, used for sequence alignment [3]. The File format is simply defined as a flat text file with one or more entries consisting of one line with a "`>`" symbol followed by a unique identifying definition line, or "defline", and one or more lines of sequence data. Here is an example. Consider a plain text file called `sequences.fa` with the following as its only content:

```
>a
ACGCGTACGTGACGACGATCG
>b
ATTTCGCGACTCTGCCTACGCTAC
>c
GGGAAACCTTTTTTT
```

The requirement that the file is "plain text", or without formatting and with a limited character set, is important. All too often, beginners to bioinformatics store sequence data in richer formats such as a word processing document. These types of files have fonts, font sizes, and other formatting markup that you can't see when viewing in the word processing application. Therefore, these files are best dealt with in text editors like the ones mentioned above. To view a FASTA file from the command line without editing it, try the applications `more` and `less` with commands like:

```
$ less sequences.fa
```

To exit from `less` simply type `q` for "quit". There aren't really any restrictions on the sequence of the defline in the standard format, except that the defline should immediately follow the "`>`" without any intervening spaces. Any kind of sequence or set of sequences can be put into a FASTA file; all 3 billion base pairs in the complete human genome, a collection of protein sequences with zinc finger domains, or the DNA sequence of the promoter of your favorite mouse gene are all valid examples.

We can use methods found in existing libraries as part of Biopython to read the contents of a FASTA file pretty easily. For example, the following code

```
from Bio import SeqIO

fastaFile = "sequences.fa"
sequences = SeqIO.parse(open(fastaFile),'fasta')
```

will read the contents of the file `sequences.fa` and store it into the object `sequences`. Now that we've read in the sequences, let's try and print them back out. To do this, we will need to introduce a "`for`" loop.

```
from Bio import SeqIO

fastaFile = "sequences.fa"
sequences = SeqIO.parse(open(fastaFile),'fasta')
for record in sequences:
        print record.id, record.seq
```

You'll notice here that we have also introduced another object called "`record`" here, and each sequence in our FASTA file is stored into such an object. Calling the methods "`id`" and "`seq`" for the `record` returns the define text and a `Seq` object. Now, let's get back to the "for loop". This is our first "control statement", or a programming statement that controls when a command is to be executed. In general, a "for loop" defines a sequence of actions to be performed. In this case, the sequence of actions is to print the `id` and `seq` for each entry in the FASTA file to standard out (print to the screen). Two other relevant pieces are the fact that the for loop statement has a colon at the end of the line, and the subsequent lines defined within the loop are indented. These are both required in python, but other programming languages may not have this restriction.

Exercise 3

Write a Biopython script that reads in a FASTA file, and prints a new FASTA file with the reverse complement of each sequence.

1.3.2 Fastq

Another very common sequence file format is "FASTQ", which is commonly used in reporting data in high-throughput sequencing experiments. Each entry of the FASTQ file format begins with the "@" symbol, followed by a unique sequence identifier. Often this identifier encodes information about the machine and sequencing run from which this data was produced. This is followed by the corresponding read sequence, and then either just the "+" symbol, or the "+" symbol followed by the same unique identifier. Then finally there is a quality string. The quality string is the exact same length as the read sequence, and each character of it encodes a quality score. Here is an example of a single entry from a FASTQ file:

```
@SRR993731.910 HWI-ST880:148:D1F64ACXX:2:1101:18790:2441 length=51
TGTCCTCTGGCTCCAGGTCTCATGATGAAAAAATTTATGGAGTCCTGGACA
+SRR993731.910 HWI-ST880:148:D1F64ACXX:2:1101:18790:2441 length=51
;?<D;,2BA=3A+2AE;<<A<EEF>E@F<FFIA?DDCD<<D>D;D9B9?##
```

The quality score is a character-encoded PHRED Score, defined as

$$S_{PHRED} = -10\log(p_{err}) \tag{1.1}$$

where p_{err} is the probability of a base-call error for that position [4]. This probability is computed by software as part of the base-calling pipeline during sequencing. Therefore, a high PHRED Score corresponds to a low probability of error. The value of the PHRED score S_{PHRED}corresponds to the ascii value of the character Q_{char} minus 33. In other words,

$$S_{PHRED} = \texttt{ord}(Q_{char}) - 33 \tag{1.2}$$

where `ord()` is a python function that returns the ASCII value of an input ASCII character. Similarly, the function `chr()` will return an ASCII character corresponding to the input ASCII value. The reason for the -33 is because the ASCII characters prior to 33 are non-printable characters, such as "space", "tab", and "return". Therfore, subtracting 33 sets the first printable character's value to 0.

The PHRED score is our first example of how probabilities are used to define scoring systems in bioinformatics. In many ways, probabilistically defined scoring systems are preferred because it makes the scores more easily interpretable, and suggests a formalism with which to compute them.

We can read a FASTQ file much the same way tha twe read in FASTA files. The `SeqIO.parse` method is able to parse this format with essentially the same command as above, but with the string "FASTQ" as the second parameter to specify file format:

```
from Bio import SeqIO
FASTQ = "reads.FASTQ"                          # FASTQ file name
data = SeqIO.parse(FASTQ,"FASTQ")   # parse the FASTQ file
```

We expect the object `data` to be a list containing many records from the FASTQ file, which can be quite large. These records can be retrieved with the following, such as the for loop:

```
>>> from Bio import SeqIO
>>> FASTQ = "reads.FASTQ"
>>> data = SeqIO.parse(FASTQ,"FASTQ")
>>> for record in data:
...     print record
...
ID: DB775P1:316:C4AGUACXX:2:1101:1748:1985
Name: DB775P1:316:C4AGUACXX:2:1101:1748:1985
Description: DB775P1:316:C4AGUACXX:2:1101:1748:1985 1:N:0:CAGATC
Number of features: 0
Per letter annotation for: phred_quality
Seq('TTTTTGTTGGGAANCTTGTGAGATTTTTGTAAATGATCGCAGTCACTTGNGCCT...GAT', SingleLetterAlphabet())
ID: DB775P1:316:C4AGUACXX:2:1101:1864:1989
Name: DB775P1:316:C4AGUACXX:2:1101:1864:1989
Description: DB775P1:316:C4AGUACXX:2:1101:1864:1989 1:N:0:CAGATC
Number of features: 0
Per letter annotation for: phred_quality
Seq('CATACACAACATANATTTGCTCATTTAGTTCCTCAAGGAACACCCGCTANTCTT...ACC', SingleLetterAlphabet())
ID: DB775P1:316:C4AGUACXX:2:1101:2323:1990
Name: DB775P1:316:C4AGUACXX:2:1101:2323:1990
Description: DB775P1:316:C4AGUACXX:2:1101:2323:1990 1:N:0:CAGATC
Number of features: 0
Per letter annotation for: phred_quality
Seq('TATGGACTACGCCGTCGAGACGGCTCACTTTGGTCTGTTCTTTAACATGNGCCA...ACG', SingleLetterAlphabet())
```

We can see that the function `phred_quality` will return the PHRED quality score for each letter/character. This is returned as a list of integers, and we'll evaluate that in a moment. To get the `Seq` object from each of the these records, we can use the `record.seq` method:

```
>>> data = SeqIO.parse(FASTQ,"FASTQ")
>>> for record in data:
...     print record.seq
...
TTTTTGTTGGGAANCTTGTGAGATTTTTGTAAATGATCGCAGTCACTTGNGCCTTCAGTTGNANTCTCGATTNATNTGGAAGGTTTCAGCC
CATACACAACATANATTTGCTCATTTAGTTCCTCAAGGAACACCCGCTANTCTTATACCTTNTNTAGTATGTTTTNAAACTATTAGAAATA
TATGGACTACGCCGTCGAGACGGCTCACTTTGGTCTGTTCTTTAACATGNGCCAGTGCTGCNGNGCAGGATCTCGNACTTTCGTGGAGGAC
```

Consider the task of computing the average error probability, p_{err}, from Equation 1.1 as a function of position. We'll need to rearrange 1.1 and compute p_{err} in terms of the Q_{char}.

$$p_{err} = 10^{-(\texttt{ord}(Q_{char})-33)/10} \tag{1.3}$$

One major "gotcha" when computing this kind of quantity is integer division. For python 2.7, plugging this equation into the python terminal will round the result, but in python 3, this issue is fixed. For example, let's try two ways of computing the probability in python:

```
>>> 10**(-(ord("b")-33)/10)
1e-07
>>> 10**(-(ord("b")-33)/10.0)
3.162277660168379e-07
```

In the first case, we are dividing the exponent by `10`, and the result is rounded. In the second case, we are using `10.0`, and we get the result using floating point arithmetic, which provides the desired answer.

Fortunately, the method `record.phred_quality` returns a list of the values of S_{PHRED}, as if they were computed from 1.2. Therefore, we have the following

$$p_{err} = 10^{-S_{PHRED}/10} \tag{1.4}$$

In practice, we'll have to open the FASTQ file as above, and loop through the records, and we'll need to store the probabilities to a list. Before we jump into that, let's consider a simple example. Here is some basic python code to work with a list object. We can declare a list, in our case a list of probabilities, by the command `p = []`. Next, we can add items to our list with the command `p.append()` with a value in the parentheses.

```
>>> p = []
>>> p.append(0.01)
>>> p.append(0.001)
>>> p.append(0.005)
>>> print p
[0.01, 0.001, 0.005]
>>> print sum(p)/len(p)
0.00533333333333
```

As you can see, we append three values to our list of probabilities, and then print the average probability, by simply computing the arithmetic mean. Let's put all these ideas together, along with a new function `plot`, that will plot a curve for us when given some values:

```
import sys
from Bio import SeqIO
import matplotlib as mpl
mpl.use('Agg')
import matplotlib.pyplot as pyplot

# this section takes care of reading in data from user
if len(sys.argv) != 2 or "-h" in sys.argv or "--help" in sys.argv:
    print >> sys.stderr, "Usage: printAverageQualityScores.py <FASTQ file>"
    sys.exit()

# read in the FASTQ file name
FASTQ = sys.argv[1]
# parse the FASTQ file
data = SeqIO.parse(FASTQ,"FASTQ")

# initialize a list of probabilities
sum_p = [0] * 200
N = [0] * 200

for record in data:
    for i,Q in enumerate(record.letter_annotations["phred_quality"]):
        # convert the PHRED score to a probability
        p_err = 10**(-float(Q)/10.0)
        # append this specific probabity to array for this sequence
        sum_p[i] += p_err
        N[i] += 1

# now for plotting. Initialize x and y arrays to plot:
x = []
y = []

# add the average probabilties to the y values:
for i in range(len(N)):
    if N[i] > 0:
        pAvg = sum_p[i]/N[i]
        x.append(i)
        y.append(pAvg)

# plot the x and y values:
pyplot.plot(x,y)
pyplot.xlabel('position (nt)')
```

```
pyplot.ylabel('Averge error probability')
pyplot.savefig('quality.png')
```

The result of running this script on some sample data, in this case the reads for some ChIP-Seq data for the Heat Shock Factor (HSF) in *Drosophila melanogaster* [5]. Note that running this script can take a long time for typical high-throughput sequencing experiments as they can be quite large. This result suggest that the probability of an error increases as the position gets larger. To some degree, errors in sequencing accumulate as once proceeds across the sequence. Repeated occurrences of the same character, or "homopolymers", are a common source of errors in high-throughput sequencing.

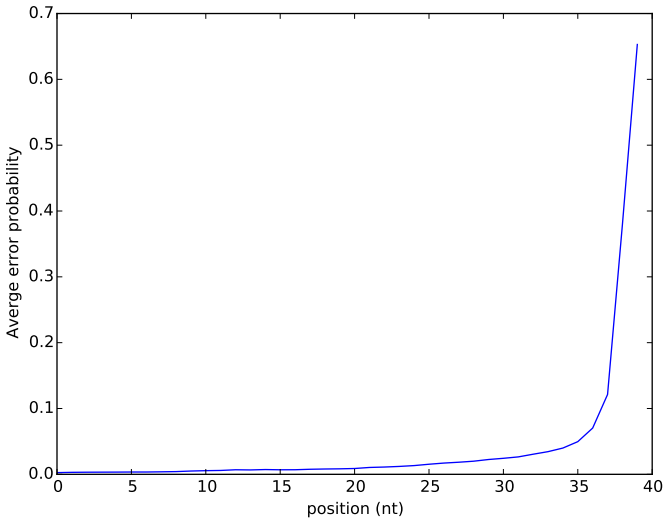

Figure 1.2: The average error probability as a function of position for ChIP-Seq data for Heat Shock Factor (HSF) in *Drosophila melanogaster*.

Exercise 4

What probability of error p_{err} corresponds to the following quality score characters:

- `!`
- `A`
- `a`
- `%`

1.3.3 GNU/Linux and Sequence Files

Another useful GNU/Linux command is `wc` (short for "word count") to count the number of lines in a file. If a FASTA file was created such that it is one line per sequence, such as the example `sequences.fasta` given above in section 1.3.1, then it is easy to count the number of sequences in the file. For example,

```
$ wc sequences.fasta
 6  6 72 sequences.fasta
```

has returned three numbers. The first number is the number of lines in the file, which is 6. The second and third numbers are the number of words and number of characters, respectively. For a FASTA file with the restriction that each sequence occupies only one line in the file, then the number of sequences is the number of lines divided by 2. Similarly, the number of sequences in a FASTQ file are the number of lines divided by 4. For the more general FASTA file, when a sequence could in theory occupy any number of lines in the file, we could count with the deflines' ">" symbol. We can grab out those lines with the program grep, which works by printing matches to a pattern with the syntax "`grep <pattern> <file>`". So we could use the ">" symbol as our pattern, and then pipe it into "wc" with a command like:

```
grep ">" sequences.fasta | wc
     3       3       9
```

This command produces the number of sequence entries in the file as the first number in the output (the 3). The pipe "|" symbol sends the output of one program into the input of another program, as with this example we are sending the output of grep into `wc` for counting. Word of warning: be careful to ensure that the ">" symbol is enclosed in quotes in this command. Failure to include the quotes will result in over-writing the file, because of the meaning of the ">" symbol on the command line. Normally, a ">" symbol on the command line redirects the output of a program to a file, and over-writes the file. For example, we could even re-direct the output of the previous example like so:

```
grep ">" sequences.fasta | wc > numSequences.txt
```

thereby printing nothing to the screen, but storing the information to a file. Here is a table summarizing all the GNU/Linux commands discussed in this chapter.

ls	list the files in the current directory
ls -a	list all files including hidden files
ls -l	long formatted list of files
cd dir	change directory to dir
cd	change to home
pwd	show current directory
mkdir dir	create a directory dir
rm file	delete file
cp file1 file2	copy file1 to file2
mv file1 file2	rename or move file1 to file2 if file2 is a directory, move file1 into directory file2
cat file	print the contents of a file to the screen
more file	output the contents of file
less file	output the contents of file
head file	output the first 10 lines of file
tail file	output the last 10 lines of file
tail -f file	output the contents of file as it grows, starting with the last 10 lines
grep pattern file	print the lines matching pattern in the file.
wc -l file	print the number of lines in a file.

1.4 Biological Sequence Databases

There are a tremendous number of sequence databases online today. New databases are published every month. In fact, there are journals devoted to databases! In what follows we will present some of the most commonly used databases for biological sequences. In later chapters, we will introduce databases that are relevant to that chapter.

Each of these databases have an associated webpage allowing data to be downloaded. Although these are useful, they are frequently updated and any book or notes on these interfaces would be quickly out of date.

Furthermore, most people can probably easily figure out how to use their interfaces effectively. Therefore, we'll focus on how to make use of Biopython to write scripts to download data from these resources on the command line, and learn what differentiates these databases.

1.4.1 NCBI

The National Center for Biotechnology Information isn't a single database, but rather a very large national resource for biomedical and genomic information [6]. Their website can be accessed at `http://www.ncbi.nlm.nih.gov/` and consists of a tremendous amount of biological sequence data in various forms from full genomes, to proteins, to single nucleotide polymorphisms (SNPs), to uploaded high-throughput sequence data.

Genbank

Genbank contains most of the world's known DNA, RNA, and protein sequences, and stores bibliographic information for the sequences as well [7]. The current release of Genbank, as of October 2015, contains $188,372,017$ sequences, comprising $202,237,081,559$ nucleotides. You can access Genbank entries from their main page `http://www.ncbi.nlm.nih.gov/genbank/` or through NCBI Nucleotide at `http://www.ncbi.nlm.nih.gov/nucc` Genbank is archival in nature, so it can contain redundant entries.

Genbank is a database, but it is also a very detailed file format. We can use the Biopython module `Entrez` to retrieve a file in Genbank format. If we already know a gi accession number for the gene, then we can retrieve the data directly from NCBI. Here is an example, with portion of the output (the full Genbank file is much too large to present here).

```
>>> from Bio import Entrez
>>> Entrez.email = "example@oregonstate.edu"
>>> p_handle = Entrez.efetch(db="protein", id='4507341', rettype="gb", retmode="text")
>>> print p_handle.read()
LOCUS       NP_003173               129 aa          DNA_input  linear   PRI 28-NOV-2015
DEFINITION  protachykinin-1 isoform beta precursor [Homo sapiens].
ACCESSION   NP_003173
VERSION     NP_003173.1  GI:4507341
DBSOURCE    REFSEQ: accession NM_003182.2
KEYWORDS    RefSeq.
SOURCE      Homo sapiens (human)
  ORGANISM  Homo sapiens
            Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi;
            Mammalia; Eutheria; Euarchontoglires; Primates; Haplorrhini;
            Catarrhini; Hominidae; Homo.
REFERENCE   1  (residues 1 to 129)
  AUTHORS   Agaeva,G.A., Agaeva,U.T. and Godjaev,N.M.
  TITLE     [Particularities of Spatial Organization of Human Hemokinin-1 and
            Mouse/Rat Hemokinin-1 Molecules]
  JOURNAL   Biofizika 60 (3), 457-470 (2015)
   PUBMED   26349209
  REMARK    GeneRIF: The spatial structures of human, mouse, and rat
            hemokinin-1 protein isoforms have been presented.
...
FEATURES             Location/Qualifiers
     source          1..129
                     /organism="Homo sapiens"
                     /db_xref="taxon:9606"
                     /chromosome="7"
                     /map="7q21-q22"
     Protein         1..129
                     /product="protachykinin-1 isoform beta precursor"
                     /note="neuropeptide gamma; neuropeptide K; tachykinin,
                     precursor 1 (substance K, substance P, neurokinin 1,
                     neurokinin 2, neuromedin L, neurokinin alpha, neuropeptide
                     K, neuropeptide gamma); tachykinin 2; protachykinin;
                     preprotachykinin; neurokinin A; protachykinin-1; PPT"
...
     CDS             1..129
                     /gene="TAC1"
```

```
                     /gene_synonym="Hs.2563; NK2; NKNA; NPK; TAC2"
                     /coded_by="NM_003182.2:247..636"
                     /note="isoform beta precursor is encoded by transcript
                     variant beta"
                     /db_xref="CCDS:CCDS5649.1"
                     /db_xref="GeneID:6863"
                     /db_xref="HGNC:HGNC:11517"
                     /db_xref="HPRD:08876"
                     /db_xref="MIM:162320"
ORIGIN
        1 mkilvalavf flvstqlfae eiganddlny wsdwydsdqi keelpepfeh llqriarrpk
       61 pqqffglmgk rdadssiekq vallkalygh gqishkrhkt dsfvglmgkr alnsvayers
      121 amqnyerrr
//
```

Genbank format is meant to be human-readable to some extent, but the files have formatting that allows them to be parsed by software tools, such as that of Biopython.

RefSeq

RefSeq (Reference Sequence) is a comprehensive and non-redundant database of biological sequences including DNA, RNA, and proteins [8]. In addition, the proteins and nucleic acid sequences corresponding to a specific gene are linked in the database. RefSeq currently maintains approximately $100,000,000$ sequence entries, each with a unique ID associated with it. You can recognize a RefSeq ID because they all have the same format:

The IDs for model entries that may be unconfirmed are formed as follows:

- `XR_` followed by numerical digits corresponds to RNA sequences that are not messenger RNAs.
- `XM_` followed by numerical digits corresponds to messenger RNAs
- `XP_` followed by numerical digits corresponds to proteins

Known and confirmed sequence IDs

- `NR_` followed by numerical digits corresponds to RNA sequences that are not messenger RNAs.
- `NM_` followed by numerical digits corresponds to messenger RNAs
- `NP_` followed by numerical digits corresponds to proteins

We can retrieve a biological sequence for a RefSeq sequence with the Biopython module `Entrez`. This allows us to download sequences directly from NCBI, and print out a FASTA file.

```
>>> from Bio import Entrez
>>> Entrez.email = "example@oregonstate.edu"
>>> rec = Entrez.read(Entrez.esearch(db="protein", term="NP_003173"))
>>> print rec['IdList']
['4507341']
>>> p_handle = Entrez.efetch(db="protein", id=rec["IdList"][0], rettype="fasta")
>>> print p_handle.read()
>gi|4507341|ref|NP_003173.1| protachykinin-1 isoform beta precursor [Homo sapiens]
MKILVALAVFFLVSTQLFAEEIGANDDLNYWSDWYDSDQIKEELPEPFEHLLQRIARRPKPQQFFGLMGK
RDADSSIEKQVALLKALYGHGQISHKRHKTDSFVGLMGKRALNSVAYERSAMQNYERRR
```

Other Databases at NCBI

NCBI is a tremendous resource with numerous useful databases. Here are a few that may be useful:

- Entrez - GQuery NCBI Global Cross-database search
- Gene - Gene is a database of genes that integrates many species. The records include nomenclature, RefSeq, maps, pathways, variations, phenotypes, links to genomes

- UniGene - An attempt to computationally identify unique transcripts from the same locus, and analyze expression by tissue, age, heath status and a number of factors. It also reports related proteins and clone resources.
- GEO - Gene Expression Omnibus. A huge database of curated gene expression data, and high-throughput sequence data.
- OMIM - Comprehensive compendium of heritable traits. Includes genetic phenotypes, and allelic variants if they have been identified. This database is also associated with "morbidMap", which identifies traits associated with diseases.
- Taxonomy - A curated database of classification and nomenclature for all organisms in the sequence database . According to NCBI, this represents 10% of the species on the planet.

1.4.2 Ensembl

The Ensembl database is a European database of biological sequences and other data. This database is produced by the European Bioinformatics Institute (EBI), which is part of the European Molecular Biology Laboratory (EMBL). This databae can be accessed at `http://ensembl.org`, and provides a huge database of biological data, in many ways similar to NCBI/GenBank. Ensembl also provides BioMart, which is a user friendly interface to retrieve genes, transcripts, and protein sequences.

1.4.3 UCSC Genome Bioinformatics

The University of California at Santa Cruz (UCSC) has a long history in genomics and bioinformatics research. Their website and associated databases called "UCSC Genome Bioinformatics" is a tremendous resource for data and tools for doing genomics, primarily on animal models systems. Their website at `http://genome.ucsc.edu/` has a genome browser, download page, and software such as BLAT for rapid alignment of sequences. BLAT results are integrated into the genome browser for alignment visualization.

1.4.4 Uniprot

The Universal Protein Resource (UniProt, `www.uniprot.org`) collects and provides a thorough database of protein sequences. UniProt is a collaboration between EMBL, the Swiss Institute of Bioinformatics, and the Protein Information Resource (PIR). UniProt actually consists of multiple databases. For example, UniRef is a database of clustered sets of sequences at varying levels of sequence identity. UniRef100 is a collection of sequences that are identical and at least 11 or more residues in length. This set comprises $70,511,308$ such clusters as of right now. Similarly, UniRef90 contains clusters that are at least 90% identical and contains $38,203,400$ clusters.

Exercise 5

What are the differences between $RefSeq$ and $GenBank$? Name at least two.

Chapter 2

Sequence Motifs

2.1 Introduction to Motifs

A biological motif, broadly speaking, is a pattern found occurring in a set of biological sequences, such as in DNA or protein sequences. A motif could be an exact sequence, such as `TGACGTCA`, or it could be a degenerate consensus sequence, allowing for ambiguous characters, such as `R` for `A` or `G`. Motifs can also be described by a probabilistic model, such as a position-specific scoring matrix (PSSM) or weight matrix.

2.2 String Matching

Frequently we want to search for an exact string or pattern within a larger sequence. For example, when using a restriction enzyme to cut a larger sequence at sequence-specific sites that match a particular pattern, we would like know where this pattern occurs in the larger sequence. This task can be called string matching. There are numerous algorithms that have been developed to make this task more efficient, such as the Knuth-Morris-Pratt algorithm [9] and the Burrows-Wheeler transform [10]. These approaches are beyond the scope of this course, but definitely worth mentioning. Let's examine how to utilize the Biopython function `nt_search` as part of the `SeqUtils` module. We can use the function as follows to search for the short pattern `ACG`.

```
>>> from Bio.Seq import Seq
>>> from Bio import SeqUtils
>>> pattern = Seq("ACG")
>>> sequence = Seq("ATGCGCGACGGCGTGATCAGCTTATAGCCGTACGACTGCTGCAACGTGACTGAT")
>>> results = SeqUtils.nt_search(str(sequence),pattern)
>>> print results
['ACG', 7, 31, 43]
```

You'll note that the function takes in two arguments. The first is the sequence to search, but it is not the `Seq` object, but the basic python string. The second argument is the pattern or string that we are searching, but this argument clearly can be the `Seq` object, which it is in this example.

Typically, we consider the DNA sequence that we are searching as double stranded, hence we want to search the forward strand and its reverse complement. In many cases for bioinformatics, such as searching an entire chromosome, it is easier to reverse complement the pattern rather that the sequence to search. In this example, this looks like

```
>>> results_rc = SeqUtils.nt_search(str(sequence),pattern.reverse_complement())
>>> print results_rc
['CGT', 11, 28, 44]
```

In this example, we have searched the forward strand of the DNA sequence. Alternatively, a biologist might want to know where the patterns occurs with positions defined along the reverse complement of the

Symbol	Meaning	Mnemonic
R	A, G	puRine
Y	C, T	pYrimidine
W	A, T	Weak (weaker basepairs, fewer hydrogen bonds)
S	G, C	Strong (stronger basepairs, more hydrogen bonds)
K	G or T	Keto (both have a keto group)
M	A or C	aMine (both have a amine group)
B	C, G, T	not A (B comes after A)
D	A, G, T	not C (D comes after C)
H	A, C, T	not G (H comes after G)
V	A, C, G	not T or U (V comes after T and U)
N	A, C, G, T	aNy base

Table 2.1: IUPAC codes for nucleotides. In this table, everywhere that T applies, U applies as well.

sequence we are searching, or the reverse strand if the DNA is double stranded. In the example of the restriction enzyme, positions defined along the reverse complement of the larger sequence are more useful for predicting the length of the resulting fragments after treatment by the restriction enzyme. This is performed as follows.

```
>>> results_rc = SeqUtils.nt_search(str(sequence.reverse_complement()),pattern)
>>> print results_rc
['ACG', 7, 23, 40]
```

The results are clearly different in terms of the positions, but they represent the same information. Because the larger sequence has a length of 54nt, we can see that we can transform between the two sets of position with the following equation:

$$\texttt{pos1} = \texttt{len(sequence)} - \texttt{pos2} - \texttt{len(pattern)} \tag{2.1}$$

Where `pos1` is the position resulting from searching the reverse complement of the pattern over the forward strand of the sequence, and `pos2` is the position of searching the pattern over the reverse strand of the sequence.

Exercise 6

Find the number of occurrences and locations of `ACTT` within the following sequnce:

```
>sequence
AGCGATCTAGCATACTTATACGCGCGCAGCTATCGATCACTTGTGCTAGTAAAGTGCGCGCCGCA
TTAAAGTGCTAGCTAGCTACTTAGCTAGCTAGTCG
```

2.3 Consensus Sequences

A consensus sequence is a string of either nucleotide or protein characters along with "degenerate characters", which specify a subset of characters. These degenerate characters can act as "wild cards", such as `N`, which can refer to any character, and are summarized in Table 2.1 [11]. They can also specify a more specific subset, such as `Y`, which specifies the pyrimidines `C` and `T`.

We can combine the standard nucleotides and IUPAC codes to form a "consensus sequence" describing a motif. For example, "Downstream Promoter Element" (DPE) in *Drosophila* occurs near position +28 from the TSS of many genes, and has the consensus sequence RGWYV [12].

2.3.1 Searching Consensus Sequences with Biopython

```
>>> from Bio import SeqUtils
>>> consensus = "RGWYV"
>>> sequence = "CGTAGCTAGCTCAGAGCAGGGACACGTGCTAGCAACAGCGCT"
>>> SeqUtils.nt_search(sequence,consensus)
['[AG]G[AT][CT][ACG]', 19]
```

2.4 Motif Finding

2.4.1 Sequence Complexity

Before we can start looking for motifs, we'll need to consider things are frequently occur in biological sequence datasets, in particular DNA sequences. The most frequent K-mer in the human genome is "AAAAAAAA", also known as poly(A). Such a sequence can be called a "low complexity" sequence, and along with simple repeats, are commonly occurring sequences that can confound motif finding and sequence alignment. Sequences like "ATATATATAT" are also frequently occurring in virtually any sequence dataset, and would be discovered by a motif search if not filtered out.

The Wooton-Federhen complexity is a score that quantifies the complexity of a sequence [13]. Put simply, the WF complexity quantifies the number of possible sequences that could be generated using the same number of As n_A, number of Cs n_C, number of Gs n_G and number of Ts n_T. This can be computed from a multi-nomial coefficient, which finds itself in a log in the equation:

$$C_{WF} = \frac{1}{N} \log_D \left(\frac{N!}{n_A! n_C! n_G! n_T!} \right)$$

Note that the log() is base $D = 4$, which is the size of the alphabet. A protein complexity score can be computed in an analogous fashion using base 20 for amino acids. The factor of $\frac{1}{N}$, where N is the number of characters in the sequence, is to ensure the value is between 0 and 1.

There is a program called `dust` (R. Tatusov and D.J. Lipman unpublished) that can mask sequence of low complexity. It can be run by specifying the sequence and a threshold.

```
dust sequences.fasta 30
```

Here we are specifying a threshold of 30, where the default is 20. Clearly, `dust` is specifying a complexity score different from WF complexity because it can be greater than 1.

2.4.2 Weight Matrices

Weight matrices are the most general representation of a motif. It is a probabilistic model that we will see can be used to compute the log-likelihood of a string being an instance of the motif compared to a "background" model.

Probabilistic Models of Motifs

The concept of the Position Specific Scoring Matrix (PSSM), also known as a weight matrix, was developed by Stormo et al [14].

Let's begin by first by defining what isn't a motif, using what is called a background model. A background model is a probabilistic representation of what a typical sequence looks like. The simplest background model is defined by single nucleotide probabilities p_A, p_C, p_G, p_T. Under such a model, the probability of a sequence is computed as the product of the individual frequencies in the sequence. This can be expressed as:

$$P(x|R) = \prod_{i=1}^{|x|} p_{x[i]} \tag{2.2}$$

Consider the nucleotide frequencies $\{p_b\}$ for the human genome. These frequencies vary from position to position, and from chromosome to chromosome. Moreover, these frequencies will vary when comparing the promoters of genes and intergenic regions.

Our motif can be described by a matrix of probabilities $f_{i,b}$. For a motif of length K, we have a matrix like:

$$f = \begin{pmatrix} f_{1A} & f_{1C} & f_{1G} & f_{1T} \\ f_{2A} & f_{2C} & f_{2G} & f_{2T} \\ f_{3A} & f_{3C} & f_{3G} & f_{3T} \\ \cdots & \cdots & \cdots & \cdots \\ f_{KA} & f_{KC} & f_{KG} & f_{KT} \end{pmatrix}$$

With this matrix, we can compute the probability (likelihood) of the sequence x given that it is an instance of the motif M by

$$P(x|M) = \prod_{i=1}^{K} f_{i,x[i]} \tag{2.3}$$

Entropy and Information Content

One helpful way of describing such a model is the Shannon entropy [15]. Shannon entropy is a measure of the uncertainty of a model, in the sense of how unpredictable a sequence generated from such a model would be. For the single-nucleotide background model, the entropy is

$$H = -\sum_{b=A}^{T} p_b \log_2 p_b$$

Note that while Shannon entropy is typically denoted H, this is not to be confused with enthalpy, which is also represented with H. The entropy is maximized when each nucleotide is equally likely, that is if $p_b = \frac{1}{4}$ for all $b \in \{A, C, G, T\}$. It is intuitive that such a model would have the highest uncertainty, for example, compared to a model where $p_A = 0.9$ and all other frequencies very low. Therefore, the maximum entropy of our background model is:

$$H_{max} = -\left(\frac{1}{4}\log_2\left(\frac{1}{4}\right) + \frac{1}{4}\log_2\left(\frac{1}{4}\right) + \frac{1}{4}\log_2\left(\frac{1}{4}\right) + \frac{1}{4}\log_2\left(\frac{1}{4}\right)\right)$$

Since $\log_2 \frac{1}{4} = -2$, we have

$$H_{max} = 2$$

When the logarithms are base 2, the units for such a quantity is called "bits", as is with BLAST scores (See Section 3.3). When using natural logs, the units are "nits". We can think of this value of 2 bits as the information content associated with knowing a particular nucleotide. A bit of information can also be understood as the number of questions necessary to unambiguously determine an unknown nucleotide. You could ask, "Is it a purine?" If the answer is "no", you could then ask is it C? The answer to the second question always guarantees, non-canonical nucleotides aside, the nucleotide's identity.

We can then compute the entropy at each position i of our motif's probability matrix by the expression

$$H_i = -\sum_{b=A}^{T} f_{i,b} \log_2 f_{i,b}$$

The Information Content of a motif at each position can be defined as the reduction in entropy. That is, the the motif provides information inasmuch as it reduces the uncertainty compared to the background model. If the change in entropy is $\Delta H_i = H_i - H_{max}$, then the information content at position i is

$$R_i = -\Delta H = H_{max} - H_i$$

Exercise 7

What is the entropy of a set of sequences with the nucleotide frequencies given by $p_A = p_T = 0.3$ and $p_C = p_G = 0.2$?

2.4.3 Relative Entropy

Another useful concept for quantifying the information in a motif is the "relative entropy", which is also known as the Kullback-Leibler divergence and as "information gain" [16]. For biological motifs, the expression for relative entropy at a particular position i is defined by the equation

$$D(f_i||p) = \sum_{b=A}^{T} f_{ib} \log\left(\frac{f_{ib}}{p_b}\right) \tag{2.4}$$

This expression quantifies how different two discrete probability distributions are, in this case the background frequencies p_b and one position of our motifs's probability matrix f_{ib}

2.4.4 Building a Weight Matrix

A weight matrix can be built or defined when one has a collection of sequences that are determined to be instances of the motif. It is essential that the motif's instances be precisely aligned, usually without gaps, such that each position of each sequence corresponds to the same position of the motif. Consider a set of sequences $S = \{x_1, x_2, x_3, ..., x_N\}$. We consider that each sequence x_j corresponds to an instances of our motif. For example, the following are instances of the binding site for the *Drosophila* transcription factor Giant:

$$\begin{aligned} x_1 &= TTTATGTGAT \\ x_2 &= GTTACGCAAT \\ x_3 &= TTAATATAAC \\ x_4 &= GTTACATAAT \\ x_5 &= CCGGCGTATT \\ &\cdots \\ x_N &= GTTACGTAAT \end{aligned}$$

One useful contract is the Kronecker delta function. For this application, consider the expression $\delta_{a,b}$, which returns 1 when $a = b$, and 0 otherwise. For example, we can check if a particular nucleotide i in sequence j is equal to the nucleotide b with the statement $\delta_{b,x_j[i]}$. Under this representation, this function is defined as:

$$\delta_{b,x_j[i]} = \begin{cases} 1 & \text{if } x_j[i] = b \\ 0 & \text{if } x_j[i] \neq b \end{cases} \tag{2.5}$$

This function is useful for connecting sequences to equations. For example, we can generate a matrix of counts C, such that the elements C_{ib} contain the number of occurrences of nucleotide b at position i in instances of our motif:

$$C_{ib} = \sum_{j=1}^{N} \delta_{b,x_j[i]}$$

This count matrix can then be normalized to represent the frequency of each nucleotide at each position by

$$f_{ib} = \frac{C_{ib}}{\sum_{b=A}^{T} C_{ib}}$$

Then, using equations 2.2 and 2.3 we can define a score as the log likelihood ratio of the probabilities of being an instance of the motif to being a random sequence. The expression of this score for a particular sequence x_j is

$$S(x_j) = \log\left(\frac{P(x|M)}{P(x|R)}\right) = \sum_{i=1}^{K} \log\left(\frac{f_{ix_j[i]}}{p_{x_j[i]}}\right)$$

As another example of the utility of the indicator matrix, this score can be represented as

$$S(x_j) = \sum_{i=1}^{K} \delta_{b,x_j[i]} \log\left(\frac{f_{ib}}{p_b}\right) = \sum_{i=1}^{\ell} \sum_{b=A}^{T} \delta_{b,x_j[i]} W_{ib}$$

In the second part of the equation, we have defined the weight matrix W with terms given by

$$W_{ib} = \log\left(\frac{f_{ib}}{p_b}\right)$$

So, for a particular sequence, $x_j = CGTAAGGT$, this equation would pick out the appropriate terms necessary to compute the score

$$S(x_j) = W_{1C} + W_{2G} + W_{3T} + W_{4A} + W_{5A} + W_{6G} + W_{7G} + W_{8T}$$

Note that for this score to make sense, you need to test a sequence x_j that is the same length as the motif.

2.4.5 Biopython Motifs

The `motifs` module

Now let's see how we can go about building a weight matrix using Biopython. First, we'll need to import modules as usual. Next, we'll need to specify a set of instances of our motif. In theory, this would be best done as reading in a FASTA file of instances. Next, we can create our motif from the list of instances. Here's how this looks:

```
>>> from Bio import motifs
>>> from Bio.Seq import Seq
>>> instances = [Seq("CGAC"),Seq("CGGG"),Seq("CGGT"),Seq("CGAT"),Seq("CGGC"),Seq("CGAG")]
>>> motif = motifs.create(instances)
>>> print motif.degenerate_consensus
CGRB
```

In this example, we have created a motif from a simple list of instances, each of which are `Seq` objects, typed in using the `motifs.create()` method. The method `degenerate_consensus` is useful for creating a concise sequence representation. We lose some information by only keeping this consensus sequence, and not keeping the matrix. By creating this motif object, we can also print out the count matrix for this motif:

```
>>> print motif.counts
       0      1      2      3
A:   0.00   0.00   3.00   0.00
C:   6.00   0.00   0.00   2.00
G:   0.00   6.00   3.00   2.00
T:   0.00   0.00   0.00   2.00
```

JASPAR sites

We often want to create a motif from external data sources. One such database of motifs is JASPAR (`http://jaspar.genereg.net`) [17]. On this webpage we can browse through various motifs. For example, in the "JASPAR CORE Insecta" section, we can see the motif MA0447.1 for the Giant binding motif (`http://jaspar.genereg.net/cgi-bin/jaspar_db.pl?ID=MA0447.1&rm=present&collection=CORE`). After downloading the "sites" file as a FASTA file in the lower left, we can see from the file "MA0447.1.sites" that it is similar to FASTA, but contains some additional information:

```
>MA0447.1        gt        1
tttctgttttggcgtaTTTATGTGATgc
>MA0447.1        gt        2
ggtggcactaccctGTTACGCAATat
>MA0447.1        gt        3
tTTAATATAACgcttctatctttgttta
>MA0447.1        gt        4
gttgttacgcgtGTTACATAATgcttcg
>MA0447.1        gt        5
aaccactgtaaagctCCGGCGTATTggc
...
```

We can see that there is additional information encoded in the capitalized letters, indicating where the binding site is. Secondly, this format doesn't work for most programs as a FASTA file because the string directly after the ">" is not unique for each line, but instead the unique information comes after with the numbering. We therefore need a special method to read in this information. The `motifs` module has a method for reading this information. Let's create a logo while we're at it:

```
>>> from Bio import motifs
>>> motif = motifs.read(open("MA0447.1.sites"),"sites")
>>> print motif.counts
        0      1      2      3      4      5      6      7      8      9
A:   9.00   0.00   1.00  29.00   0.00   5.00   0.00  30.00  33.00   0.00
C:   4.00   1.00   0.00   0.00  29.00   0.00   3.00   2.00   1.00   8.00
G:  18.00   1.00   2.00   4.00   0.00  30.00   1.00   1.00   0.00   4.00
T:   4.00  33.00  32.00   2.00   6.00   0.00  31.00   2.00   1.00  23.00

>>> motif.weblogo("giant_LOGO.pdf",format="pdf")
```

This last command produces a LOGO image and requires an internet connection. It actually sends the data to the website `http://weblogo.berkeley.edu/` [18]. The resulting LOGO image looks like this, which is similar to the motif logo seen on the JASPAR database:

We've already seen how to score a particular sequence with a weight matrix, and how to build a weight matrix form a collection of instances. In practice, we often have longer sequences where we don't know precisely where the instances are, but know that the sequences contain subsequences that are instances of the motif.

2.4.6 Gibbs Sampling

Gibbs sampling has many applications in statistical physics in general [19]. Is sometimes called a Markov-chain Monte Carlo (MCMC) approach. Briefly, Gibbs sampling starts with a set of sequences in which there is a motif of unknown sequence content. First, random initial positions of the motif are selected. Next, a weight matrix is built from those instances. The weight matrix is then used to re-score the positions of the input sequences, to find a new optimal set of instances, and the weight matrix is subsequently updated. This process is repeated until convergence. Perhaps surprisingly, this method is capable of finding motifs quite well.

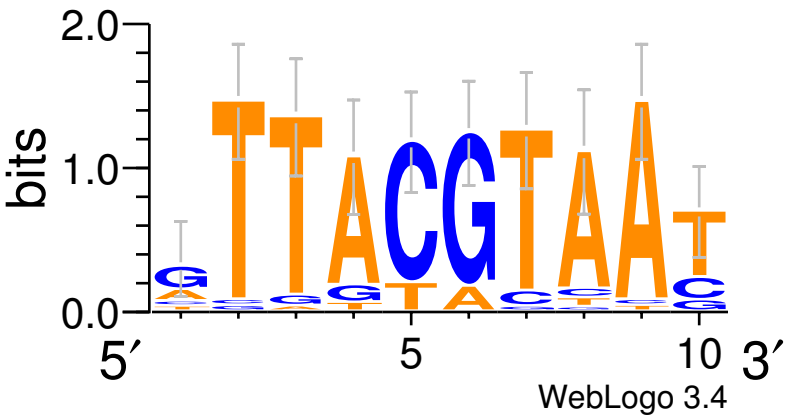

Figure 2.1: Sequence LOGO created with weblogo for the giant motif

2.4.7 MEME and the EM Algorithm

One of the most widely used software tools for motif discovery is MEME: Multiple EM for Motif Elicitation. MEME uses Expectation Maximization to find the best binding sites [20]. The algorithm computes the parameters that maximize the expected value of the (log-)likelihood pf the model. The EM algorithm attempts to learn the "missing data", in this case the positions of the instances of the motif. Once the instances of the motif are identified, an updated weight matrix can be computed.

MEME starts with some initialization of motifs built from the enriched K-mers. Essentially this list of initial motifs is achieved through a heuristic method, which applies one iteration of EM on all K-mers. The algorithm computes an expression for the expected value of the log-likelihood. It then computes the value of the motif membership variable Z that maximizes the expected value of the log-likelihood. This process is repeated until convergence.

There are numerous options for running MEME. For example, one can select how the instances of the motifs are distributed in the input sequences. They are specified by the "-mod" option and can be:

1. oops - Exactly one instance per sequence.
2. zoops - One or zero instances per sequence.
3. anr - Any number of instances per sequence.

To find a motif in the input file "sequences.fa" that has a maximum motif length (specified with the -maxw option) and requiring exactly one occurence per sequence, we would use the following command

```
meme sequences.fa -dna -mod oops -maxw 12
```

Similarly, to find 2 different motifs, each with a maximum length (width) of 10, and such that each sequence has zero or one instance, we would use the following command:

```
meme sequences.fa -dna -mod zoops -maxw 10 -nmotifs 2
```

For a full list of options, try typing `meme --help` on the command line.

Chapter 3

Sequence Alignments

Biological sequences evolve through a process of mutation and natural selection. By comparing two sequences, we can determine whether two sequences have a common evolutionary origin if their similarity is unlikely to be due to chance. Before we get into how this is done, we must also consider that there are many types of evolutionary relationships among sequences.

similarity: the degree of resemblance between two sequences.

identity: the state of possessing the same subsequence. One often quantifies the percent identity between two sequences.

homology: the state of sharing a common evolutionary origin.

orthology: The state of being homologous sequences that arose from a common ancestral gene during speciation.

paralogy: The state of being homologous sequences that arose from a common ancestral gene from gene duplication.

Sequence alignment is the process of arranging the characters of a pair of sequences such that the number of matched characters is maximized. We can describe the alignment between two sequences with the following notation:

```
GCGTAACACGTGCG--
 |  ||| ||||||
AC--AACCCGTGCGAC
```

The vertical bars "|", or pipes, represent matching characters. Gaps, indicated by the dash "-" are inserted in between characters in place of missing characters to optimize the number of matches. It is critical that sequence alignments are viewed in a monospace font, such as Courier, so that the width of characters don't offset the alignment.

3.1 Alignment Algorithms and Dynamic Programming

One of the first attempts to align two sequences was carried out by Vladimir Levenstein in 1965, called "edit distance", and now is often called Levenshtein Distance. The edit distance is defined as the number of single character edits necessary to change one word to another. Initially, he described written texts and words, but this method was later applied to biological sequences. One of the most commonly used algorithms for computing the edit distance is the Wagner-Fischer algorithm, a Dynamic Programming algorithm.

Dynamic Programming optimally phrases the full problem as the optimal solution to the smaller pieces (sub-problems). The overall problem can then be expressed as a composition of the sub-problems. In addition to the Wagner-Fischer algorithm, numerous other dynamic programming algorithms have been developed for aligning biological sequences including the Needleman-Wunsch and Smith-Waterman Algorithms.

3.1.1 Needleman-Wunsch Algorithm

The Needleman-Wunsch Algorithm is a global alignment algorithm, meaning the result always aligns the entire input sequences [24]. We've just defined a scoring matrix in section 8.3, for protein alignment, but for nucleotide sequences, we often use a simpler scoring matrix such as

$$S_{a,b} = \begin{cases} 1, & \text{if } a = b \\ -1, & \text{if } a \neq b \end{cases} \tag{3.1}$$

In addition to a scoring matrix, we also need to define penalties for gaps. The most common gap penalty is the linear gap penalty, defined as

$$c_L(d) = Gd,$$

which is just proportional to the length d of the gap by a parameter $G < 0$. A more complicated approach is an "affine gap penalty", which penalizes opening a gap by one parameter, and extending the gap by another parameter. For example, such a gap penalty can by defined by

$$c_A(d) = G + (d-1)E$$

which includes a gap open parameter G and a gap extension parameter E. In practice, an affine gap penalty is much more difficult to compute.

Dynamic programming for sequence alignments begins by defining a matrix or a table, to compute the scores. For example, let's consider aligning the nucleotide sequences $x = \texttt{CAGCTAGCG}$ and $y = \texttt{CCATACGA}$. For Needleman-Wunsch, let's define a matrix F, such that the terms $F_{i,j}$ correspond to the score of aligning the subsequences $x[1..i]$ and $y[1..j]$. We proceed from the upper left of this matrix at $F_{0,0}$, and fill in the matrix as we move from left to right and from top to bottom. Here the rows of F will correspond to the positions of x, and the columns will correspond to the positions of y.

When computing the terms of the matrix F, we need to define a set of boundary conditions, namely that the score at the boundaries is associated with the penalty all the way up to that position. This is achieved by setting $F_{i,0} = i \times G$ and $F_{0,j} = j \times G$ for $1 \leq i \leq |x|$ and $1 \leq j \leq |y|$.

We compute the terms of the matrix F using a "recurrence relation", such that the terms of a given cell of the matrix F are defined in terms of the neighboring cells. Needleman-Wunsch uses the following recurrence relation:

$$F_{i,j} = max \begin{cases} F_{i-1,j} + G & \text{skip a position of } x \\ F_{i,j-1} + G & \text{skip a position of } y \\ F_{i-1,j-1} + S_{x[i],y[j]} & \text{match/mismatch} \end{cases} \tag{3.2}$$

Let's consider the result of computing the matrix F using the scoring matrix in 3.1, and using a linear gap penalty $G = -1$. The result is presented in Table 3.1.1. In this matrix, each term then corresponds to the score up to the character at that i and j position of the sequences x and y respectively. The rows will correspond to positions i in the sequence x, and the columns will correspond to positions j of y.

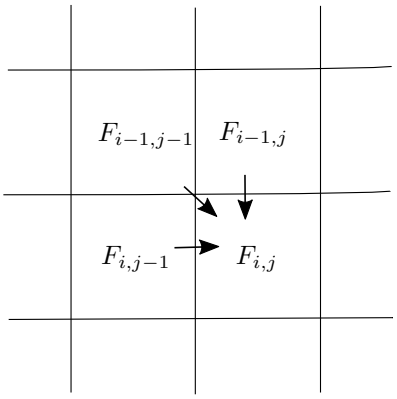

Figure 3.1: Each term of the matrix is computed using Equation 3.2 from the term above, to the left, and diagonally above-left.

The terms F_{ij} of the matrix F can be filled out as is done with the following matrix, with each cell computed using the recursion relation in Equation 3.2, as depicted in Figure 3.1.1. The optimal path is shown in blue.

	-	C	C	A	T	A	C	G	A
-	0	-1	-2	-3	-4	-5	-6	-7	-8
C	-1	1	0	-1	-2	-3	-4	-5	-6
A	-2	0	0	1	0	-1	-2	-3	-4
G	-3	-1	-1	0	0	-1	-2	-1	-2
C	-4	-2	0	-1	-1	-1	0	-1	-2
T	-5	-3	-1	-1	0	-1	-1	-1	-2
A	-6	-4	-2	0	-1	1	0	-1	0
G	-7	-5	-3	-1	-1	0	0	1	0
C	-8	-6	-4	-2	-2	-1	1	0	0
G	-9	-7	-5	-3	-3	-2	0	2	1

Table 3.1: A dynamic programming matrix to compute the score for Needleman-Wunsch Alignment.

In addition to the F matrix, it is common to keep track of a traceback matrix T, that keeps track of from where each term was computed from, in other words the maximum term in Eq 3.2. Table 3.1.1 demonstrates such a traceback matrix. One key complication is dealing with ties. One strategy is to favor adjacent matched characters as much as possible; therefore, we would favor diagonal terms before above or to the left.

	-	C	C	A	T	A	C	G	A
-	·	·	·	·	·	·	·	·	·
C	·	×	↖	←	←	←	↖	←	←
A	·	↑	↖	↖	←	↖	←	←	↖
G	·	↑	↖	↑	↖	↖	↖	↖	←
C	·	↖	↖	←	↖	↖	↖	←	↖
T	·	↑	↑	↖	↖	←	↑	↖	↖
A	·	↑	↑	↖	←	↖	←	←	↖
G	·	↑	↑	↑	↖	↑	↖	↖	←
C	·	↖	↖	↑	↖	↑	↖	←	↖
G	·	↑	↑	↑	↖	↑	↑	↖	←

Table 3.2: A traceback matrix for Needleman-Wunsch.

Finally, we have the result of the alignment. Here is the result of the Needleman-Wunsch alignment.

Because it is a global alignment, the full sequence is included and the alignment ends on the first and last positions. There are, however, gaps at the first and last positions as this example illustrates.

```
-CAGCTAGCG-
  ||   ||  ||
CCA--TA-CGA
```

3.1.2 Smith-Waterman

In most applications we are only interested in aligning a small portion of the sequence to produce a local alignment. Furthermore, we don't necessarily want to force the first and last residues to be aligned. Smith-Waterman is an alignment algorithm that has these properties.

We can define a set of boundary conditions for the scoring matrix $F_{i,j}$, namely that the score is 0 at the boundaries so that $F_{i,0} = F_{0,j} = 0$ for $1 \leq i \leq |x|$ and $1 \leq j \leq |y|$. Define the recurrence relation:

$$F_{i,j} = max \begin{cases} F_{i-1,j} + G & \text{skip a position of } x \\ F_{i,j-1} + G & \text{skip a position of } y \\ F_{i-1,j-1} + S_{x[i],y[j]} & \text{match/mismatch} \\ 0 & \text{zero-out negative scores} \end{cases} \tag{3.3}$$

In addition to the different boundary conditions, a key difference between Needleman-Wunsch (global alignment) and Smith-Waterman (local alignment) is that whereas with the global alignment we start tracing back from the lower right term of the matrix, for the local alignment we start at the maximum value. This value corresponds to the last matched character of the optimal alignment.

	-	C	C	A	T	A	C	G	A
-	0	0	0	0	0	0	0	0	0
C	0	1	1	0	0	0	1	0	0
A	0	0	0	2	1	1	0	0	1
G	0	0	0	1	1	0	0	1	0
C	0	1	1	0	0	0	1	0	0
T	0	0	0	0	1	0	0	0	0
A	0	0	0	1	0	2	1	0	1
G	0	0	0	0	0	1	1	2	1
C	0	1	1	0	0	0	2	1	1
G	0	0	0	0	0	0	1	3	2

Table 3.3: A matrix for Smith-Waterman, with an optimal path labeled in blue.

The optimal score corresponds to the 3 in the last row, but second to last column. The optimal path results in an alignment with four matching positions. The traceback matrix can be built while computing the alignment matrix, and all paths are haulted when a score of zero is reached.

	-	C	C	A	T	A	C	G	A
-	·	·	·	·	·	·	·	·	·
C	·	↖	↖	←	·	·	↖	←	·
A	·	↑	↖	↖	←	↖	↑	↖	↖
G	·	·	·	↑	↖	↖	↖	↖	↑
C	·	↖	↖	↑	↖	↖	↖	↑	↖
T	·	↑	↖	↖	×	←	↑	↖	·
A	·	·	·	↖	↑	↖	←	←	↖
G	·	·	·	↑	↖	↑	↖	↖	←
C	·	↖	↖	←	·	↑	↖	↑	↖
G	·	↑	↖	↖	·	·	↑	↖	←

Table 3.4: A traceback matrix for Smith-Waterman

For Smith-Waterman, we typically report just the sub-aligment corresponding to the positive scores. We can report an alignment consisting of just the two sequences.

```
TAGCG
|| ||
TA-CG
```

3.1.3 Comparison

Although there are some similarities, there are a couple of key differences between Needleman-Wunsch and Smith-Waterman Algorithms. Here is a summary:

Needleman-Wunsch Algorithm

1. Computes the optimal **global alignment** in O(nm)
2. Backtracking begins in lower right: global alignment
3. Allows negative scores

Smith-Waterman Algorithm

1. Computes optimal **local alignment** in O(nm)
2. Backtracking begins at largest value (not necessarily lower right)
3. Negative scores are zeroed out

3.1.4 Aligning DNA vs Proteins

When performing sequence alignments it is important to realize some of the key differences between aligning nucleic acid sequences and aligning protein sequences. We've seen that proteins can have more sophisticated substitution matrices, such as BLOSUM and PAM, that incorporate probabilistic models. That said, in Chapter 4 we will get into some probabilistic models of nucleotide substitution that could be incorporated into a scoring system. By building substitution matrices from curated alignments that record evolutionary changes that occur in nature, the protein substitution matrices encode the chemical similarity between amino acids. For example, scores are better for substituting between two polar amino acids compared to mutating from polar to non-polar. Furthermore, when inside the coding region of a gene, the third position of codons is more mutable because this position can typically change without changing the amino acid that it encodes.

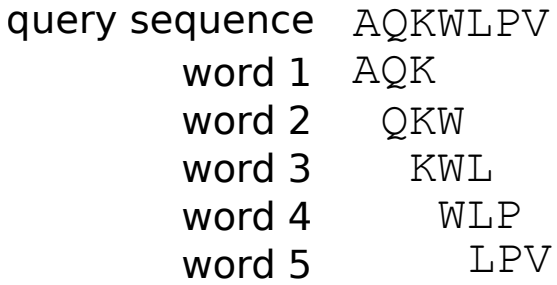

Figure 3.2: Make K-mer word list of the query sequence (Proteins often $K = 3$)

3.2 Alignment Software

3.2.1 BLAST: Basic Local Alignment Search Tool

The BLAST algorithm (Basic Local Alignment Search Tool) developed by Altschul (1990) combines indexing of a database of sequences, and heuristics to approximate Smith-Waterman alignment, but is $50\times$ faster. The approach of BLAST is to index a search database using K-mers, subsequences of length K, for each of the sequences in the database. A query sequence is input to the program to search for similar sequences in the database. After low-complexity sequences are removed, all K-mers of the query sequence are listed, and possible matches in the database are identified that would have an alignment score as good as T, a predefined score threshold. The matching K-mers are extended into stretches of matching K-mers, that are called High-scoring Segment Pairs (HSPs), resulting in matches that are longer than K. Two or more of these HSPs are combined to form a longer alignment. Ultimately Smith-Waterman alignment is performed on just these strongly matching sequences, and this is what is reported.

In summary, the approach is as follows:

1. Remove low-complexity regions or sequence repeats in the query sequence.
2. Make K-mer word list of the query sequence (Proteins often $K = 3$)
3. List the possible 20^3 matching words with a scoring matrix
4. Reduce the list of word matches with threshold T
5. Extend the exact matches to High-scoring Segment Pairs (HSPs)
6. List all HSPs and evaluate significance
7. Combine two or more HSPs into a longer alignment
8. Report the gapped Smith-Waterman local alignments of the query and each of the matched database sequences.

3.3 Alignment Statistics

When evaluating a BLAST score, it is important to have a statistical framework for evaluating the significance of a "BLAST hit". Here we present such a system where we consider our score S as a random variable. Because BLAST identifies the maximum scoring alignment, we can describe the cumulative distribution of BLAST scores with the Generalized Extreme Value (GEV) distribution:

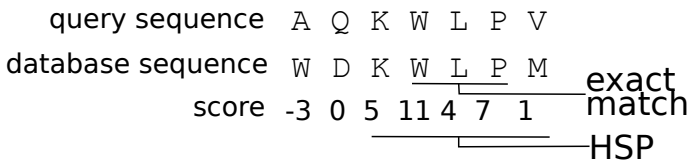

Figure 3.3: K-mers that match with a score above T are extended form High-scoring Segment Pairs (HSPs).

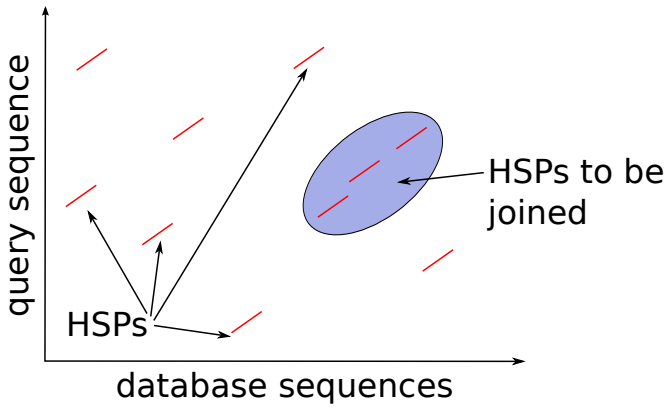

Figure 3.4: Extend the exact matches to High-scoring Segment Pairs (HSPs)

$$P(S \leq x) = \exp\left(-e^{-\lambda(x-u)}\right)$$

The parameter u is the location parameter of the GEV, and is expressed here in terms of the length n of the query sequence, and the length m of the entire database. K here is a constant that is particular to and computed from a particular database.

$$u = \frac{\ln Knm}{\lambda}$$

The p-value is the probability of a score greater than or equal to S due to chance, and is given by:

$$P(S \geq x) = 1 - \exp\left(-Knme^{-\lambda x}\right)$$

This equation comes from the Poisson distribution. If we define E-value (expected number of hits at this score or greater due to chance) as:

$$E = Knme^{-\lambda S}$$

The p-value can then be simplified as:

$$P(S \geq x) = 1 - e^{-E}$$

After a linear transformation, the score S' can be computed in terms of bits.

$$S' = \frac{\lambda S - \ln K}{\ln 2}$$

The updated equation for E-value is much simpler

$$E = nm \times 2^{-S'}$$

3.3.1 Running BLAST from the command line

BLAST can be run on the command line pretty easily. To do this, you need a sequence, or set of sequences to align, and a database to align to. First, let's create the database to align to. This can be created using a FASTA file of sequences. For example, if we have the FASTA file for the human genome `hg38.fa`, we can format the database with `formatdb` using the following command:

```
formatdb -p F -t hg38 -n hg38 -i hg38.fa
```

In this command, the `-p F` command indicates that this is a nucleotide sequence, and not a protein sequence. Specifically, the `-p` specifies protein, and the `F` says that this is "false", specifying that the input data is not protein. The second two commands give the database the title and name "`hg38`". The name specified by the `-n` command provides a basename for the output files used in the database, and also gives a label to be used when referring to the database in BLAST. Finally, the `-i` command specifies the input file, which is the FASTA file for the genome.

Next, we can run BLAST using the command `blastall`. This tool allows you to run different versions of BLAST, specified by the `-p` command. To run a nucleotide query against a nucleotide database, we use `blastn`. The full command is as follows:

```
blastall -p blastn -i sequences.fa -d hg38  -o sequences_hg38_blast.txt
```

Here we specify the input sequences, the query, with the `-i` command. Then we specify the database that we are aligning to, using the `-d` flag, referring to the database that we just created with `formatdb`. Finally, we specify an output file to write the results to, using the `-o` flag.

3.4 Short Read Mapping

The growth of high-throughput sequencing has led to a parallel growth of software applications for rapidly aligning short reads. Although BLAST was designed for fast alignment, these new tools are even faster for the alignment of short sequence reads. We will discuss these methods further in chapter 9

Chapter 4

Multiple Sequence Alignments, Molecular Evolution, and Phylogenetics

4.1 Multiple Sequence Alignment

A multiple sequence alignment is an alignment of more than 2 sequences. It turns out that this makes the problem of alignment much more complicated, and much more computationally expensive. Dynamic programming algorithm such as Smith-Waterman can be extended to higher dimensions, but at a significant computing cost. Therefore, numerous methods have been developed to make this task faster.

4.1.1 MSA Methods

There have been numerous methods developed for computing MSAs do make them more computationally feasible.

Dynamic Programming

Despite the computational cost of MSA by Dynamic Programming, there have been approaches to compute multiple sequence alignments using these approaches. The programs MSA and MULTALIN use dynamic programming. This process takes $O(L^N)$ computations for aligning N sequences of length L. The Carrillo-Lipman Algorithm uses pairwise alignments to constrain the search space. By only considering regions of the multi-sequence alignment space that are within a score threshold for each pair of sequences, the L^N search space can be reduced.

Progressive alignments

Progressive alignments begin by performing pairwise alignments, by aligning each pair of sequences. Then it combines each pair and integrates them into a multiple sequence alignment. The different methods differ in their strategy to combine them into an overall multiple sequence alignment. Most of these methods are "greedy", in that they combine the most similar pairs first, and proceed by fitting the less similar pairs into the MSA. The following programs use progressive alignment:

1. T-Coffee
2. ClustalW
3. PSAlign

Iterative Alignment

Iterative Alignment is another approach that improves upon the progressive alignment because it starts with a progressive alignment and then iterates to incrementally improve the alignment with each iteration. The following programs use iterative alignment:

1. CHAOS/DIALIGN
2. MUSCLE

Multiple Genome Alignments

Specialized multiple sequence alignment approaches ahve been developed for aligning complete genomes, to overcome the challenges associated with aligning such long sequences. The following programs have been developed for aligning full genomes:

1. MLAGAN (using LAGAN)
2. MULTIZ (using BLASTZ)
3. MUSCLE
4. MUMmer

4.1.2 MSA File Formats

There are several file formats that are specifically designed for multiple sequence alignment. These approaches can differ in their readability by a human, or are can be designed for storing large sequence alignments.

Multi-Fasta Format: mfa

Probably the simplest multiple sequence alignment format is the Multi-FASTA format (mfa), which is essentially like a FASTA file, such that each sequence provides the alignment sequence (with gaps) for a given species. The deflines can in some cases only contain information about the species, and the file name, for example, could contain information about what sequence is being described by the file. For short sequences the mfa can be human readable, but for very long sequences it can become difficult to read. Here is an example `.mfa` file that shows the alignment of a small (28aa) *Drosophila melanogaster* peptide called *Sarcolamban (isoform C)* with its best hits to `nr`.

```
>D.melanogaster
--------------------------------------------------
--------------------------------MSEARNLFTTFGILAILL
FFLYLIYA--------------------VL---------------
>D.sechellia
--------------------------------------------------
--------------------------------MSEARNLFTTFGILAILL
FFLYLIYAPAAKSESIKMNEAKSLFTTFLILAFLLFLLYAFYEAAF
>D.pseudoobscura
MSEAKNLMTTFGILAFLLFCLYLIYASNNSKRWPTFCGEAEFRSENSESQ
LLRAFSYERLEQCPNKKYPPKQPTTTTTKPIKMNEARSLFTTFLILAFLL
FLLYAFYEA--------------------AF---------------
>D.busckii
--------------------------------------------------
--------------------------------MNEAKSLVTTFLILAFLL
FLLYAFYEA--------------------AF---------------
```

Clustal

The Clustal format was developed for the program `CLUSTAL W`, but has been widely used by many other programs. This file format is intended to be fairly human readable in that it expresses only a fixed length of the alignment in each section, or block. Here is what the Clustal format looks like for the same *Sarcolamban* example:

```
CLUSTAL W (1.83) multiple sequence alignment


D.melanogaster      ------------------------------------------------------------
D.sechellia         ------------------------------------------------------------
D.pseudoobscura     MSEAKNLMTTFGILAFLLFCLYLIYASNNSKRWPTFCGEAEFRSENSESQLLRAFSYERL
D.busckii           ------------------------------------------------------------


D.melanogaster      ----------------------MSEARNLFTTFGILAILLFFLYLIYA------------
D.sechellia         ----------------------MSEARNLFTTFGILAILLFFLYLIYAPAAKSESIKMNE
D.pseudoobscura     EQCPNKKYPPKQPTTTTTKPIKMNEARSLFTTFLILAFLLFLLYAFYEA-----------
D.busckii           ----------------------MNEAKSLVTTFLILAFLLFLLYAFYEA-----------
                                           *.**:.*.*** ***:***:** :*

D.melanogaster      ----------VL----------------
D.sechellia         AKSLFTTFLILAFLLFLLYAFYEAAF
D.pseudoobscura     ----------AF----------------
D.busckii           ----------AF----------------
                             :
```

The clustal format has the following requirements, which can make it difficult to create one manually. First, the first line in the file must start with the words "`CLUSTAL W`" or "`CLUSTALW`". Other information in the first line is ignored, but can contain information about the version of `CLUSTAL W` that was used to create it. Next, there must be one or more empty lines before the actual sequence data begins. The rest of the file consists of one or more blocks of sequence data. Each block consists of one line for each sequence in the alignment. Each line consists of the sequence name, defline, or identifier, some amount white space, then up to 60 sequence symbols such as characters or gaps. Optionally, the line can be followed by white space followed by a cumulative count of residues for the sequences. The amount of white space between the identifier and the sequences is usually chosen so that the sequence data is aligned within the sequence block. After the sequence lines, there can be a line showing the degree of conservation for the columns of the alignment in this block. Finally, all this can be followed by some amount of empty lines.

MAF

The Multiple Alignment Format (MAF) can be a useful format for storing multiple sequence alignment information. It is often used to store full-genome alignments at the UCSC Genome Bioinformatics site. The file begins with a header beginning with `##maf` and information about the version and scoring system. The rest of the file consists of alignment blocks. Alignment blocks start with a line that begins with the letter `a` and a score for the alignment block. Each subsequent line begins with either an `s`, a `i`, or an `e` indicating what kind of line it is. The lines beginning with `s` contain sequence information. Lines that begin with `i` typically follow each `s`-line, and contain information about what is occurring before and after the sequences in this alignment block for the species considered in the line. Lines beginning with `e` contain information about empty parts of the alignment block, for species that do not have sequences aligning to this block. For example, the following is a portion of the alignment of the Human Genome (GRCh38/hg38) `chr22` with 99 vertebrates.

```
##maf version=1 scoring=roast.v3.3
a score=49441.000000
```

```
s hg38.chr22              10514742 28 +  50818468 acagaatggattattggaacagaataga
s panTro4.chrUn_GL393523    96163 28 +    405060 agacaatggattagtggaacagaagaga
i panTro4.chrUn_GL393523 C 0 C 0
s ponAbe2.chrUn           66608224 28 -  72422247 aaagaatggattagtggaacagaataga
i ponAbe2.chrUn           C 0 C 0
s nomLeu3.chr6            67506008 28 - 121039945 acagaatagattagtggaacagaataga
i nomLeu3.chr6            C 0 C 0
s rheMac3.chr7            24251349 14 + 170124641 --------------tggaacagaataga
i rheMac3.chr7            C 0 C 0
s macFas5.chr7            24018429 14 + 171882078 --------------tggaacagaataga
i macFas5.chr7            C 0 C 0
s chlSab2.chr26           21952261 14 -  58131712 --------------tggaacagaataga
i chlSab2.chr26           C 0 C 0
s calJac3.chr10           24187336 28 + 132174527 acagaatagaccagtggatcagaataga
i calJac3.chr10           C 0 C 0
s saiBol1.JH378136        10582894 28 -  21366645 acataatagactagtggatcagaataga
i saiBol1.JH378136        C 0 C 0
s eptFus1.JH977629        13032669 12 +  23049436 ----------------gaacaaagcaga
i eptFus1.JH977629        C 0 C 0
e odoRosDiv1.KB229735       169922 2861 +    556676 I
e felCat8.chrB3           91175386 3552 - 148068395 I
e otoGar3.GL873530          132194  0 +  36342412 C
e speTri2.JH393281         9424515 97 +  41493964 I
e myoLuc2.GL429790         1333875  0 -  11218282 C
e myoDav1.KB110799          133834  0 +   1195772 C
e pteAle1.KB031042        11269154 1770 -  35143243 I
e musFur1.GL896926        13230044 2877 +  15480060 I
e canFam3.chr30           13413941 3281 +  40214260 I
e cerSim1.JH767728        28819459 183 +  61284144 I
e equCab2.chr1            43185635 316 - 185838109 I
e orcOrc1.KB316861        20719851 245 -  22150888 I
e camFer1.KB017752          865624 507 +   1978457 I
```

The `s` lines contain 5 fields after the `s` at the beginning of the line. First, the source of the column usually consists of a genome assembly version, and chromosome name separated by a dot ".". Next is the start position of the sequence in that assembly/chromosome. This is followed by the size of the sequence from the species, which may of course vary from species to species. The next field is a strand, with "+" or "-", indicating what strand from the species' chromosome the sequence was taken from. The next field is the size of the source, which is typically the length of the chromosome in basepairs from which the sequence was extracted. Lastly, the sequence itself is included in the alignment block.

4.2 Phylogenetic Trees

A phylogenetic tree is a representation of the evolutionary history of a character or sequence. Branching points on the tree typically represent gene duplication events or speciation events. We try to infer the evolutionary history of a sequence by computing an optimal phylogenetic tree that is consistent with the extant sequences or species that we observe.

4.2.1 Representing a Phylogenetic Tree

A phylogenetic tree is often a "binary tree" where each branch point goes from one to two branches. The junction points where the branching takes place are called "internal nodes". One way of representing a tree is with nested parentheses corresponding to branching. Consider the following example

```
((A,B),(C,D));
```

where the two characters `A` and `B` are grouped together, and the characters `C` and `D` are grouped. The semi-colon at the end is needed to make this tree in proper "newick" tree format. One of the fastest ways to draw a tree on the command line is the "ASCII tree", which we can draw by using the function `Phylo.draw_ascii()`. To use this, we'll need to save our tree to a text file that we can read in. We could read the tree as just text into the python terminal (creating a string), but that would require loading an additional module `cStringIO` to use the function `StringIO`. Therefore, it might be just as easy to save it to a text file called `tree.txt` that we can read in. Putting this together, we can draw the tree with the following commands:

```
>>> from Bio import Phylo
>>> tree = Phylo.read("tree.txt","newick")
>>> Phylo.draw_ascii(tree)
```

```
                                 ____________________________________ A
  ______________________________|
 |                              |____________________________________ B
_|
 |                               ____________________________________ C
 |______________________________|
                                |____________________________________ D
```

This particular tree has all the characters at the same level, and does not include any distance or "branch length" information. Using real biological sequences, we can compute the distances along each brach to get a more informative tree. For example, we can download 18S rRNA sequences from NCBI Nucleotide. Using clustalw, we can compute a multiple sequence alignment, and produce a phylogenetic tree. In this case, the command

```
$ clustalw -infile=18S_rRNA.fa -type=DNA -outfile=18S_rRNA.aln
```

will produce the output file `18S_rRNA.dnd`, which is a tree in newick tree format. The file contains the following information

```
(fly:0.16718,(mouse:0.00452,human:0.00351):0.05186,chicken:0.24654);
```

You'll note, this is the same format as the simple example above, but rather than a simple label for each character/sequence, there is a label and a numerical value, corresponding to the brach length, separated by a colon. In addition, each of the parentheses are followed by a colon and a numerical value. In each case, the value of the branch length corresponds to the substitutions per site required to change one sequence to another, a common unit of distance used in phylogenetic trees. This value also corresponds to the length of the branch when drawing the tree. The command to draw a tree image is simply `Phylo.draw`, which will allow the user to save the image.

```
>>> from Bio import Phylo
>>> tree = Phylo.read('18S_rRNA.dnd','newick')
>>> Phylo.draw(tree)
```

The resulting image can be seen in Figure 4.1, and visually demonstrates the branch lengths corresponding to the distance between individual sequences. The x-axis in the representation corresponds to this distance, but the y-axis only separates taxa, and the distance along the y-axis does not add to the evolutionary distance between sequences.

4.2.2 Pairwise Distances

Phylogenetic trees are often computed as binary trees with branch lengths that optimally match the pair-wise distances between species. In order to compute a phylogenetic tree, we need a way of defining this distance. One strategy developed by Feng and Doolittle is to compute a distance from the alignment scores computed

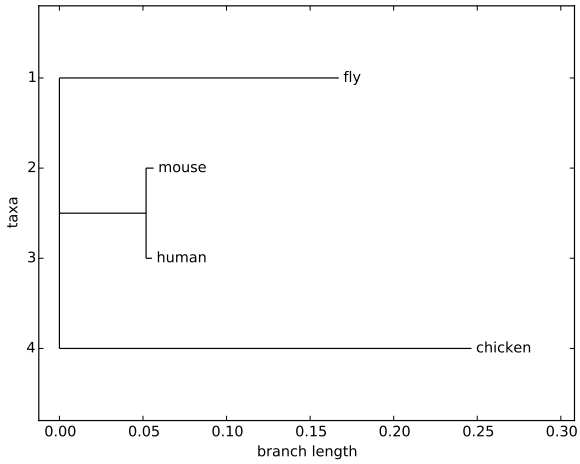

Figure 4.1: A phylogenetic tree computed with Clustalw for 18S rRNA sequences for *Drosophila melanogaster*, *Homo sapiens*, *Mus musculus*, and *Gallus gallus*

from pair-wise alignments. The distance is defined in terms of an "effective", or normalized, alignment score for each pair of species. This distance is defined as

$$D_{ij} = -\ln S_{eff}(i,j)$$

so that pairs of sequences (i,j) that have high scores will have a small distance between them. The effective score $S_{eff}(i,j)$ is defined as

$$S_{eff}(i,j) = \frac{S_{real}(i,j) - S_{rand}(i,j)}{S_{iden}(i,j) - S_{rand}(i,j)} \times 100$$

Where in this expression, $S_{real}(i,j)$ is the observed pairwise similarity between sequences from species i and j. The value $S_{iden}(i,j)$ is the average of the two scores when you align species i and j to themselves, which represents the score corresponding to aligning "identical" sequences, the maximum possible score one could get. $S_{rand}(i,j)$ is the average pairwise similarity between randomized, or shuffled, versions of the sequences from species i and j. After this normalization, the score $S_{eff}(i,j)$ ranges from 0 to 100.

4.3 Models of mutations

Evolution is a multi-faceted process. There are many forces involved in molecular evolution. The process of mutation is a major force in evolution. Mutation can happen when mistakes are made in DNA replication, just because the DNA replication machinery isn't 100% perfect. Other sources of mutation are exposure to radiation, such as UV radiation, certain chemicals can induce mutations, and viruses can induce mutations of the DNA (as well as insert genetic material). Recombination is a genetic exchange between chromosomes or regions within a chromosome. During meiosis, genes and genomic DNA are shuffled between parent chromosomes. Genetic drift is a stochastic process of changing allele frequencies over time due to random sampling of organisms. Finally, natural selection is the process where differences in phenotype can affect survival and reproduction rates of different individuals.

4.3.1 Genetic Drift

Because genetic drift is a stochastic process, it can be modeled as a "rate". The rate of nucleotide substitutions r can be expressed as

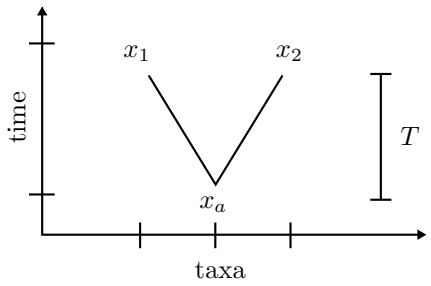

Figure 4.2: For two extant species x_1 and x_2 diverged for a time T from a common ancestor x_a, the mutation rate can be expressed as a time $2T$ separating x_1 and x_2.

$$r = \frac{\mu}{2T}$$

where μ is the substitutions per site across the genome, and T is the time of divergence of the two (extant) species to their common ancestor. The factor of two can be understood by the fact that it takes a total of $2T$ to go from x_1 to x_2, stopping at x_a along the way. The mutations that occur over time separating x_1 and x_2 can be viewed as distributed over a time $2T$.

In most organisms, the rate is observed to be about 10^{-9} to 10^{-8} mutations per generation. Some viruses have higher mutation rates 10^{-6} mutations per generation. The generation times of different species can also affect the nucleotide substitution rate r. Organisms with shorter generation times have more opportunities for meiosis per unit time.

When mutation rates are evaluated within a gene, positional dependence in nucleotide evolution is observed. Because of the degeneracy in the genetic code, the third position of most codons has a higher substitution rate. Some regions of proteins are conserved domains, hence the corresponding regions of the gene have lower mutation rates compared to other parts of the gene. Other genes such as immunoglobulins have very high mutation rates and are considered to be "hypervariable"

Noncoding RNAs have functional constraints to preserver hairpins, and may have sequence evolution that preserves base pairing through compensatory changes on the paired nucleotide. The result is that many noncoding RNAs such as tRNAs have very conserved structure, but vary at the sequence level.

4.3.2 Substitution Models

The branches of phylogenetic trees can often represent the the expected number of substitutions per site. That is, the distance along the branches of the phylogenetic tree from the ancestor to the extant species correspond to the expected number of substitutions per site in the time it takes to evolve from the ancestor to the extant species.

A substitution model describes the process of substitution from one set of characters to another through mutation. These models are often neutral, in the sense that selection is not considered, and the characters mutate in an unconstrained way. Furthermore, these models are typically considered to be independent from position to position.

Substitution models typically are described by a rate matrix Q with terms Q_{ab} that describe mutating from character a to b for terms where $a \neq b$. The diagonal terms of the matrix are defined so that the sum of the rows are zero, so that

$$Q_{aa} = -\sum_{b \neq a} Q_{ab} \tag{4.1}$$

In general, the matrix Q can be defined as:

$$Q = \begin{pmatrix} * & Q_{AC} & Q_{AG} & Q_{AT} \\ Q_{CA} & * & Q_{CG} & Q_{CT} \\ Q_{GA} & Q_{GC} & * & Q_{GT} \\ Q_{TA} & Q_{TC} & Q_{TG} & * \end{pmatrix} \tag{4.2}$$

where the diagonal terms are defined such that it is consistent with Equation 4.1. The rate matrix is associated with a probability matrix $P(t)$, which describes the probability of observing the mutation from a to b in a time t by the terms $P_{ab}(t)$. We want these probabilities to be multiplicative, meaning that $P(t_1)P(t_2) = P(t_1 + t_2)$. The mutations associated with amounts of time t_1 and t_2 applied successively can be understood as the same as the mutations associated with $t_1 + t_2$. Furthermore, the derivative of the equation can be expressed as

$$P'(t) = P(t)Q \tag{4.3}$$

The solution to this equation is the exponential function. The rate matrix itself can be exponentiated to compute the probability of a particular mutation in an amount of time t, which can be computed using the Taylor series for the exponential function.

$$P(t) = e^{Qt} = \sum_{n=0}^{\infty} Q^n \frac{t^n}{n!} \tag{4.4}$$

Each such model also assumes an equilibrium frequencies π, which describe the probability of each nucleotide after the system has reached equilibrium.

4.3.3 Jukes-Cantor 1969 (JC69)

The simplest substitution model was proposed by Jukes and Cantor (JC69), and describes a constant mutation rate μ, and equilibrium frequencies such that $\pi_A = \pi_C = \pi_G = \pi_T = \frac{1}{4}$. The equilibrium frequencies describe the frequencies of each nucleotide that result after the system has evolved under this model for a "very long time". The rate matrix for the Jukes-Cantor model is then given by:

$$Q = \begin{pmatrix} -\frac{3}{4}\mu & \frac{\mu}{4} & \frac{\mu}{4} & \frac{\mu}{4} \\ \frac{\mu}{4} & -\frac{3}{4}\mu & \frac{\mu}{4} & \frac{\mu}{4} \\ \frac{\mu}{4} & \frac{\mu}{4} & -\frac{3}{4}\mu & \frac{\mu}{4} \\ \frac{\mu}{4} & \frac{\mu}{4} & \frac{\mu}{4} & -\frac{3}{4}\mu \end{pmatrix} \tag{4.5}$$

It can be shown that the full expression for computing $P(t) = e^{Qt}$ is:

$$P(t) = \begin{pmatrix} \frac{1}{4}(1+3e^{-\mu t}) & \frac{1}{4}(1-e^{-\mu t}) & \frac{1}{4}(1-e^{-\mu t}) & \frac{1}{4}(1-e^{-\mu t}) \\ \frac{1}{4}(1-e^{-\mu t}) & \frac{1}{4}(1+3e^{-\mu t}) & \frac{1}{4}(1-e^{-\mu t}) & \frac{1}{4}(1-e^{-\mu t}) \\ \frac{1}{4}(1-e^{-\mu t}) & \frac{1}{4}(1-e^{-\mu t}) & \frac{1}{4}(1+3e^{-\mu t}) & \frac{1}{4}(1-e^{-\mu t}) \\ \frac{1}{4}(1-e^{-\mu t}) & \frac{1}{4}(1-e^{-\mu t}) & \frac{1}{4}(1-e^{-\mu t}) & \frac{1}{4}(1+3e^{-\mu t}) \end{pmatrix} 00 \tag{4.6}$$

Therefore, we can solve this system to get the probabilities $P_{ab}(t)$, which can be expressed as

$$P_{ab}(t) = \begin{cases} \frac{1}{4}(1+3e^{-\mu t}) & \text{if } a = b \\ \frac{1}{4}(1-e^{-\mu t}) & \text{if } a \neq b \end{cases} \tag{4.7}$$

Finally, we note that the sum of the terms of a row of the matrix Q that correspond to mutation changes give us the expected value of the distance $\hat{d}$ in substitutions per site. For the Jukes-Cantor model, this corresponds to $\hat{d} = \frac{3}{4}\mu t$. Substituting this into Equation 4.7 for the terms involving change ($a \neq b$), gives $p = \frac{1}{4}(1 - e^{-\frac{4}{3}\hat{d}})$, which can be solved for $\hat{d}$ to give:

$$\hat{d} = -\frac{3}{4}\ln(1 - \frac{4}{3}p) \tag{4.8}$$

This formula is often called the Jukes-Cantor distance formula, and it gives a way to relate the proportion of sites that differ p to the evolutionary distance $\hat{d}$, which stands for the expected number of substitutions in the time t for a mutation rate μ. This formula corrects for the fact that the proportion of sites that differ, p, does not take into account sites that mutate, and then mutate back to the original character.

4.3.4 Kimura 1980 model (K80)

The Jukes-Cantor model considers all mutations to be equally likely. The Kimura model (K80) considers the fact that transversions, mutations involving purine to pyrimidines or vice versa, are less likely than transitions, which are from purines to purines or pyrimidines to pyrimidines. Therefore, this model has two parameters: a rate α for transitions, and a rate β for transversions.

$$Q = \begin{pmatrix} -(2\beta+\alpha) & \beta & \alpha & \beta \\ \beta & -(2\beta+\alpha) & \beta & \alpha \\ \alpha & \beta & -(2\beta+\alpha) & \beta \\ \beta & \alpha & \beta & -(2\beta+\alpha) \end{pmatrix} \tag{4.9}$$

Applying a similar derivation as was done for the JC69 model, we get the following distance formula for K80:

$$d = -\frac{1}{2}\ln(1-2p-q) - \frac{1}{4}\ln(1-2q) \tag{4.10}$$

where p is the proportion of sites that show a transition, and q is the proportion of sites that show transversions.

4.3.5 Felsenstein 1981 model (F81)

The Felsenstein model essentially makes the assumption that the rate of mutation to a given nucleotide has a specific value equal to its equilibrium frequency π_b, but these value vary from nucleotide to nucleotide. The rate matrix is then defined as:

$$Q = \begin{pmatrix} * & \pi_C & \pi_G & \pi_T \\ \pi_A & * & \pi_G & \pi_T \\ \pi_A & \pi_C & * & \pi_T \\ \pi_A & \pi_C & \pi_G & * \end{pmatrix} \tag{4.11}$$

4.3.6 The Hasegawa, Kishino and Yano model (HKY85)

The Hasegawa, Kishino and Yano model takes the K80 and F81 models a step further and distinguishes between transversions and transitions. In this expression, the transitions are weighted by an additional term κ.

$$Q = \begin{pmatrix} * & \pi_C & \kappa\pi_G & \pi_T \\ \pi_A & * & \pi_G & \kappa\pi_T \\ \kappa\pi_A & \pi_C & * & \pi_T \\ \pi_A & \kappa\pi_C & \pi_G & * \end{pmatrix} \tag{4.12}$$

4.3.7 Building Phylogenetic Trees

Now that we have a probabilistic framework with which to describe phylogenetic distances, we need some methods to build a tree from a set of pair-wise distances. Here are two basic approaches to building Phylogenetic Trees.

Unweighted Pair Group Method with Arithmetic Mean (UPGMA) Algorithm

UPGMA is a phylogenetic tree building algorithm that uses a type of hierarchical clustering. This algorithm builds a rooted tree by creating internal nodes for each pair of taxa (or internal nodes), starting with the most similar and proceeding to the least similar. This approach starts with a distance matrix d_{ij} for each taxa i and j. When branches are built connecting i and j, an internal node k is created, which corresponds to a cluster C_k containing i and j. Distances are updated such that the distance between a cluster (internal node) and a leaf node is the average distance between all members of the cluster and the leaf node. Similarly, the distance between clusters is the average distance between members of the cluster.

Neighbor Joining Algorithm

One of the issues with UPGMA is the fact that it is a greedy algorithm, and joins the closest taxa first. There are tree structures where this fails. To get around this, the Neighbor joining algorithm normalizes the distances $d_{i,j}$ using the following formula:

$$D_{i,j} = d_{i,j} - \frac{1}{n-2}\left(\sum_{k=1}^{n} d_{i,k} + \sum_{k=1}^{n} d_{j,k}\right)$$

Begin with a star tree, and a matrix of pair-wise distances d(i,j) between each pair of sequences/taxa, an updated distance matrix that normalizes compared to all distances is created. The updated distances $D_{i,j}$ are computed. The closest taxa using this distance are identified, and an internal node is created such that the distance along the branch connecting these two nearest taxa is the distance $D_{i,j}$. This process s repeated using the new (internal) node as a taxa and the distances are updated.

4.3.8 Evaluating the Quality of a Phylogenetic Tree

Maximum Parsimony

Maximum Parsimony makes the assumption that the best phylogenetic tree is that with the shortest branch lengths possible, which corresponds to the fewest mutations to explain the observable characters.

This method begins by identifying the phylogenetically informative sites. These sites would have to have a character present (no gaps) for all taxa under consideration, and not be the same character for all taxa. Then trees are constructed, and characters (or sets of characters) are defined for each internal node all the way up to the root. Then each tree has a cost defined, corresponding to the total number of mutations that need to be assumed to explain that tree. The tree with the shortest total branch length is typically chosen. The length of the tree is defined as the sum of the lengths of each individual character (or column of the alignment) L_j, and possibly using a weight w_j for different characters (often just 1 for all columns, but could weight certain positions more).

$$L = \sum_{j=1}^{C} w_j L_j$$

In this expression, the length of a character L_j can be computed as the total number of mutations needed to explain this distribution of characters given the topology of the tree.

Maximum Likelihood

Maximum likelihood is an approach that computes a likelihood for a tree, using a probabilistic model for each tree. The probabilistic model is applied to each branch of the tree, such as the Jukes-Cantor model, and the likelihood of a tree is the product of the probabilities of each branch. Generally speaking, we seek to find the tree $\mathcal{T}$ that has the greatest likelihood given the D.

$$P(\mathcal{T}|D) = \frac{P(D|\mathcal{T})P(\mathcal{T})}{P(D)}$$

These probabilities could be computed from an evolutionary model, such as Jukes-Cantor model. For example, if we have observed characters x_1 and x_2, and an unknown ancestral character x_a, and lengths of the branches from x_a to be t_1 and t_2 respectively, we could compute the likelihood of this simple tree as

$$P(x_1, x_2|\mathcal{T}, t_1, t_2) = \sum_a p_a P_{a,x_1}(t_1) P_{a,x_2}(t_2)$$

where we are summing over all possible ancestral characters a, and computing the probability of mutating along the branches using a probabilistic model. This probabilistic model can be the same terms of the probability matrices discussed in Equation 4.6

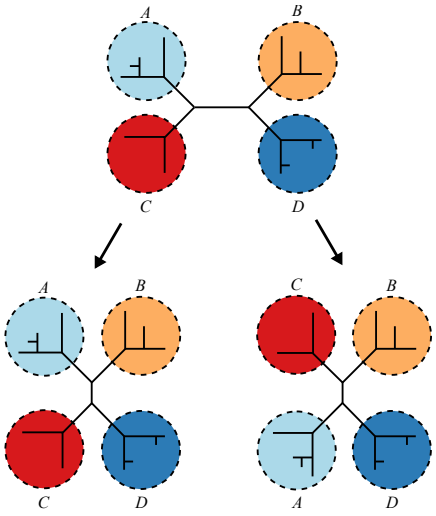

Figure 4.3: Nearest Neighbor Interchange operates by swapping out the locations of subgraphs within a tree. For a tree with four sub-trees, there are only 2 possible interchanges.

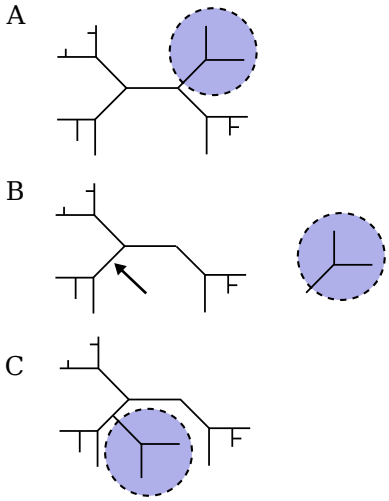

Figure 4.4: A depiction of SPR Tree searching method. **A.** One subtree of the larger tree structure is selected. **B.** An attachment point is selected. **C.** The subtree is then "grafted" or attached to the attachment point.

4.3.9 Tree Searching

In addition to computing the quality of a specific tree, we also want to find ways of searching the space of trees. Because the space of all possible trees is so large, we can't exhaustively enumerate them all in a practical amount of time. Therefore, we need to sample different trees stochastically. Two such methods are Nearest Neighbor Interchange (NNI) and Subtree Pruning and Re-grafting (SPR).

Nearest Neighbor Interchange

For a tree toplogy that contains four or more taxa, the Nearest Neighbor Interchange (NNI) exchanges subtrees within the larger tree. It can be shown that for a tree with four subtrees there are only 2 distinct ways to exchange subtrees to create a new tree. Therefore, each such application produces two new trees.

Subtree Pruning and Regrafting

The Subtree Pruning and Regrafting (SPR) method takes a subtree from a larger tree, removes it and reattaches it to another part of the tree.

Chapter 5

Genomics

In this chapter, we will learn how to work with genomic data and genome annotations and associated file formats. It is hoped that this chapter will serve as a basic introduction to genomics, with the understanding that it is a far broader field than the types of genome annotations presented here. Many of the chapters, such as the next chapter "Transcriptomics", and many before, could be considered as a subject matter as part of the field of genomics.

5.1 The Three Fundamental "Gotchas" of Genomics

Whereas many sciences have three or four fundamental laws that describe them, research in genomics often encounters exceptions to rules rather than universal principles. However, when learning genomics for the first time, researchers often encounter the same problems that cause the same strange things to happen in their data. These errors are so common for beginners in genomics, they can be thought of as the "fundamental gotchas" of genomics:

1. Different genome assemblies
2. Different chromosome deflines
3. 0 vs 1-based coordinates

5.1.1 Different Genome Assemblies/Annotations

The first fundamental gotcha involves the fact that for any model organism, there are typically many different genome assembly versions. For example, one might download two different datasets, such as a set of genomic positions, and try to compare them, only to realize that they were compared against different assemblies of the genome. In many cases, such as human genome assembly versions, each being incremental improvements in the accuracy of the assembly.

5.1.2 Different Chromosome Deflines

Depending on from where you downloaded the genome annotation, you could possibly encounter an assembly that uses one defline for "Chromosome 1", and another that uses a different one. For example, some annotations such as from UCSC Genome Bioinformatics, would use "chr1", and other databases may just use "1". You definitely need to double check these values when comparing different formats.

5.1.3 0 vs 1-based coordinates

The third fundamental gotcha involves comparing data from different file formats, and not being aware that some file formats use 0-base positions, and others use 1-based positions. Whereas 0-based coordinates consider the first position of a chromosome to be position 0, 1-based coordinates consider the first position

Format	Position system
GFF/GTF	1-based
BLAST results	1-based
BLAT results	1-based
maf	0-based
bam	0-based
sam	1-based
BED	0-based
wig	1-based

Table 5.1: Different file formats use different position systems.

of a chromosome to be position 1. In many ways, 0 is more natural for computer science, because typically programming languages use 0-based coordinates to describe strings. Biologists may be more accustomed to counting positions starting at 1. There are many different file formats for storing the positions of genomic locations, each use one of these two position systems, so it's important to know which is which, and be aware that there is a difference.

5.2 Genomic Data and File Formats

5.2.1 Formats for Genomic Locations

One of the most common form of genome annotation is that of the genome location. Often we want to annotate and label a binding site for a protein, or a gene by the genomic locations of the exons. Therefore, we need to define file formats that specify genomic locations.

Often, we can describe a genomic location by four variables that specify the "genomic coordinate". The "genomic coordinate" can be specified by a chromosome, a start position, a stop position, and a strand. Clearly, the fundamental gotchas are important when evaluating these four values. For strand, it is most common to see "+" and "-" to refer to the forward and reverse strand. However, if you look at enough databases you will find things like "F" and "R" to denote strand.

BED Files

BED files are a simple file format to describe genomic locations. The BED file was developed by UCSC Genome Bioinformatics, and is a common format found when downloading data from that site.

The first three required BED fields are:

1. chrom: The name of the chromosome (e.g. chr3, chrY) or scaffold.
2. chromStart: The starting position of the feature in the chromosome or scaffold. The first base in a chromosome is numbered 0.
3. chromEnd: The "non-inclusive" ending position of the feature in the chromosome or scaffold. The chromEnd base is not included in the display of the feature, so it behaves much like ranges and substrings in python. For example, the first 100 bases of a chromosome are defined as chromStart=0, chromEnd=100, and span the bases numbered 0-99.

So using this core required format, the BED file can be a useful streamlined representation of genomic positions where strand is not important. We can expand the file format to include other information, for example when we want to store data for gene models. The next columns we could add are:

4. name: Defines the name of the BED line. This label is displayed to the left of the BED line in the Genome Browser window when the track is open to full display mode or directly to the left of the item in pack mode.

5. score: A score between 0 and 1000.
6. strand: Defines the strand - either '+' or '-'.
7. thickStart: The starting position at which the feature is drawn thickly (for example, the start codon in gene displays).
8. thickEnd: The ending position at which the feature is drawn thickly (for example, the stop codon in gene displays).
9. itemRgb: An RGB value of the form R,G,B (e.g. 255,0,0). If the track line itemRgb attribute is set to "On", this RBG value will determine the display color of the data contained in this BED line.
10. blockCount: The number of blocks (exons) in the BED line.
11. blockSizes: A comma-separated list of the block sizes. The number of items in this list should correspond to blockCount. Exons sizes are stored here if this is gene data
12. blockStarts: A comma-separated list of block starts. All of the blockStart positions should be calculated relative to chromStart. The number of items in this list should correspond to blockCount. Exons start positions are stored.

GFF Files

GFF format is a great way to store gene annotation information. In many ways the columns of GFF are designed for genes, but by including a dot "." for the columns that don't apply to your annotation, you can also store more simple annotations such as the locations of a motif instance.

1. seqname - The name of the sequence. Must be a chromosome or scaffold.
2. source - The program that generated this feature.
3. feature - The name of this type of feature. Some examples of standard feature types are "CDS", "start_codon", "stop_codon", and "exon".
4. start - The starting position of the feature in the sequence. The first base is numbered 1.
5. end - The ending position of the feature (inclusive).
6. score - A score between 0 and 1000. If the track line useScore attribute is set to 1 for this annotation data set, the score value will determine the level of gray in which this feature is displayed (higher numbers = darker gray). If there is no score value, enter ".".
7. strand - Valid entries include '+', '-', or '.' (for don't know/don't care).
8. frame - If the feature is a coding exon, frame should be a number between 0-2 that represents the reading frame of the first base. If the feature is not a coding exon, the value should be '.'.
9. group - All lines with the same group are linked together into a single item.

GTF Files

A GTF file is very similar to a GFF file, but with a few different specifications. The first eight GTF fields are the same as GFF. The group field has been expanded into a list of attributes. Each attribute consists of a type/value pair. Attributes must end in a semi-colon, and be separated from any following attribute by exactly one space. The attribute list must begin with the two mandatory attributes:

1. gene_id value - A globally unique identifier for the genomic source of the sequence.
2. transcript_id value - A globally unique identifier for the predicted transcript.

Bam/Sam

Col	Field	Type	Brief description
1	QNAME	String	Query template NAME
2	FLAG	Int	bitwise FLAG
3	RNAME	String	Reference sequence NAME
4	POS	Int	1-based leftmost mapping POSition
5	MAPQ	Int	MAPping Quality
6	CIGAR	String	CIGAR string
7	RNEXT	String	Ref. name of the mate read
8	PNEXT	Int	Position of the mate/next read
9	TLEN	Int	observed Template LENgth
10	SEQ	String	segment SEQuence
11	QUAL	String	ASCII of Phred-scaled base QUALity+33

A bam file is essentially a binary, indexed version of sam. The program `samtools` can allow quick retrieval of reads from a genomic region. Sam files is an optional output format for many alignment algorithms, but many people prefer to convert them to bam files because of faster retrieval of reads for a particular genomic position due to indexing.

5.2.2 Quantitative Tracks

We often would like to annotate quantitative data on a genome browser; hence, it is critical to have file formats devoted to this kind of data. Consider, for example, plotting the GC content as a function of position throughout the genome.

BedGraph

```
track type=bedGraph
chrom1  chromStart1  chromEnd1  dataValue1
chrom2  chromStart2  chromEnd2  dataValue2
```

Wiggle, and BigWig

Wiggle file format is a common way to display quantitative tracks. The format is relatively simple, but takes on two different version: `variableStep`, and `fixedStep`. Each of these are specified at the top of the file. For the fixed step, we have a fixed number of bases between positions presented in the file:

```
fixedStep  chrom=chrN  start=position  step=stepInterval  [span=windowSize]
dataValue1
dataValue2
dataValue3
...
```

Because the start position is specified, and the step size is specified, positions don't need to be specified in the file. For the variable width, we'll need to specify the position for each value:

```
variableStep  chrom=chrN  [span=windowSize]
chromStart1  dataValue1
chromStart2  dataValue2
chromStart3  dataValue3
...
```

The optional parameter "span" does not include the brackets when used in practice, and here only indicates that it is optional. The span essentially indicates that a value can be specified as applying to a range of positions, starting at the given chromStart value.

5.3 Genome Browsers

Genome browsers provide an interactive way to navigate the data associated with a genome in a visual way. Much like a web browser or an interactive map application. The file formats given above are exactly the kind of files that a genome browser would read and present visually.

5.3.1 IGV

The Integrated Genome Viewer (IGV) is a powerful desktop genome browser that allows you to relatively easily add and remove tracks and modify them. IGV allows you to export to various image formats, including scalable vector formats such as SVG files. In some cases the available RAM on one's computer may be a limitation for loading too many tracks into IGV.

5.3.2 UCSC Genome Browser

The UCSC genome browser is a web-based genome browser. You can download, install, and host a version of the UCSC genome browser on your own computer, but you can also add tracks to the genome browser hosted at `https://genome.ucsc.edu/cgi-bin/hgGateway`.

5.3.3 Gbrowse

Gbrowse is probably one of the most common genome browsers out there. Many databases such as Flybase, or Wormbase have Gbrowse integrated in to the database for users to navigate the genomic data presented. Gbrowse allows for the display to be exported into multiple file formats, including scalable file formats.

5.3.4 JBrowse

Jbrowse is very similar to Gbrowse, but allows for asynchronous queries to the database, effectively making for a faster experience. One can scroll the position rapidly and have features immediately presented to the user without having to reload the page. Jbrowse allows the display to be exported in png, but not in a scalable file format.

Chapter 6

Transcriptomics

Broadly speaking, Transcriptomics is the study of transcriptomes, the sum total of all transcripts in a cell. Transcriptomics seeks to build transcriptome annotations, and to measure differential expression of transcripts from different tissue types or treatments.

6.1 High-throughput Sequencing (HTS)

High-throughput sequencing (also known as deep sequencing) is a technology that has been developed in the late 20th century and continues to improve today. High-thoughtput sequencing has many applications, and most relevant for transcriptomics is deep sequencing of RNA, called RNA-seq. The word "deep" in deep sequencing refers to the depth of sequencing, characterized by:

$$D = \frac{N \times L}{T}$$

where the depth D is computed from the number of reads N, the length of the reads L, and the size of the transcriptome T. The size of the transcriptome T can be thought of the length of the union of all transcripts for a particular system. A depth of $2\times$ means that on average a location in the genome would have 2 reads mapping to that location, assuming a uniform distribution of reads. This equation assumes that the reads are uniformly distributed, which is almost never true. Nevertheless it serves as a good approximation.

High-throughput sequencing can produce hundreds of millions of reads per sequencing lane, and in many cases the lane is multiplexed to include multiple samples per lane. This technology has enabled scientists to study biological phenomena at a genome-wide scale, and has enabled the discovery of a number of properties of transcription.

6.2 RNA-seq reads

RNA deep sequencing is a method where a cDNA library is created for an RNA sample, and is sequenced using high-throughput sequencing, producing hundreds of millions of reads. Notably, there are different types of RNA-seq data sets. First, single-end reads involve the sequencing of one read per cDNA fragment, typically in the 5' to 3' direction. Paired-end reads have two reads per fragment, with the two paired-reads called "mates". Often the first mate is sequenced in the direction of transcription, and the second mate is sequenced in the opposite 3' to 5' direction. This, however, can vary on the sequencing technology used. The manual for tophat 2 (`https://ccb.jhu.edu/software/tophat/manual.shtml`) provides the information on Table 6.1.

Typicaly with paired end data, one receives two files labeled `R1` and `R2`. The reads in each file correspond to pairs if they have the same read ID, excluding the possibility of the reads to be labeled `R1` and `R2` or possibly `1` and `2`. The different arrangements of paired-end reads explained in Table 6.1 are represented in Figure **??**. In practice, the library type can be determined by aligning paired reads from both the R1 and R2 fastq file to the genome, and examining the relative orientation of the reads and overlapping transcripts.

Library Type	Examples	Description
fr-unstranded	Standard Illumina	Reads from the left-most end of the fragment (in transcript coordinates) map to the transcript strand, and the right-most end maps to the opposite strand.
fr-firststrand	dUTP, NSR, NNSR	Same as above except we enforce the rule that the right-most end of the fragment (in transcript coordinates) is the first sequenced (or only sequenced for single-end reads). Equivalently, it is assumed that only the strand generated during first strand synthesis is sequenced.
fr-secondstrand	Ligation, Standard SOLiD	Same as above except we enforce the rule that the left-most end of the fragment (in transcript coordinates) is the first sequenced (or only sequenced for single-end reads). Equivalently, it is assumed that only the strand generated during second strand synthesis is sequenced.

Table 6.1: A table of the different library types allowed for Tophat. A similar set of types are available for Hisat.

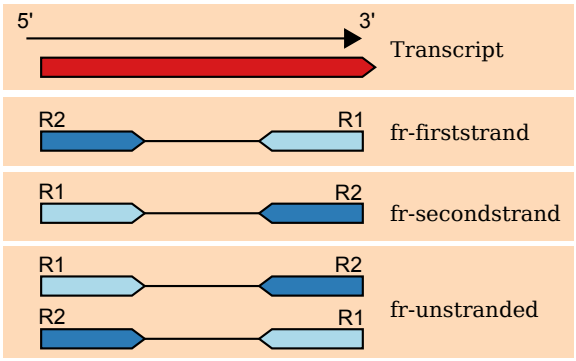

Figure 6.1: A representation of an the different library types found in RNA-Seq data. The most common is fr-secondstrand, with the `R1` read in the direction of transcription.

Aligning RNA-Seq data

The first step of an RNA-Seq analysis is aligning the reads to the genome. Actually, there are approaches to RNA-Seq without a genome discussed in section 6.2.1. For organisms with an assembled genome, the typical approach is to align the reads to the genome. There are many different programs to do this, with different features. For eukaryotic organisms, it is important that the alignment allows for gaps corresponding to introns. Another important consideration is the number of mismatches allowed in the alignment. For a highly polymorphic genome, for example, one would want to increase the number of allowed mismatches under the assumption that the transcripts sequenced could be transcribed from a slightly different genome.

As an example of aligning reads to the genome, let's consider Tophat2, which is part of the Tuxedo suite. For the other steps of the RNA-Seq analysis pipeline we will also examine parts of the tuxedo suite pipeline.

The first step of aligning is to have an index of the genome. The index of the genome, much like a BLAST index created with `formatdb` provides a quick look-up of genomic locations to assist with the alignment. In the case of Tophat2, which uses bowtie2 for the alignment, the creation of the index is based on the Burrows-wheeler transform. The index of the genome consists of multiple files, but each has the same prefix, or "file base" as an input. As an example, for the mouse genome `mm10` that has a file base `mm10`, we will

have the six files `mm10.1.bt2`, `mm10.2.bt2`, `mm10.3.bt2`, `mm10.4.bt2`, `mm10.rev.1.bt2`, `mm10.rev.2.bt2`. So, using default parameters, aligning with Tophat2 would be performed by the following command:

```
tophat -o reads_mm10_tophat mm10 reads.fastq
```

where the `-o` option species the output directory where the output files are going to go. Probably the most important of the output files in this directory is the bam file called `accepted_hits.bam`. This file contains a sorted record of the alignment information. There are other files created such as log files that could provide useful information for tracking down issues if something goes wrong. Also among these output files is `junctions.bed`, which provides a BED file of all splice junctions identified, and the number of reads that span that particular junction.

Of course, we don't always want to run Tophat2 with default parameters. For example, we might want to align with a reference annotation, such that only gaps are considered that correspond to gaps in an input GTF file. To achieve this, we would use the following command:

```
tophat -o reads_mm10_tophat --no-novel-juncs -G ensembl_mm10.gtf mm10 reads.FASTQ
```

Here we have specified to not include any novel junctions with the `--no-novel-juncs` flag. Required with this flag is to specify a GTF file with the `-G` flag. In this case we have specified an annotation from Ensembl for mm10.

We may consider adding to this a command to specify the number of mismatches (the default is 2). This can be changed with the `-N/--read-mismatches` flag.

Transcriptome Assembly

Transcriptome assembly is the process by which the aligned reads are compiled into transcript models that best represent the read data. At minimum, a transcript model must consist of reads that map to that location. Introns must be supported by a gapped alignment that spans the intronic region, typically further requiring that the donor and acceptor sites are present at the start and end of the intron (GU on the 5' end and AG on the 3' end of the intron). We may optionally want to assemble a transcriptome for an RNA-Seq experiment when we don't have a transcriptome annotation available. We may also want to perform this step when none is available. A schematic of how this process is performed is depicted in Figure 6.2

The next step of the Tuxedo suite that corresponds to transcriptome assembly is `cufflinks`. This can be run with default parameters with the following command:

```
cufflinks reads_mm10_tophat/accepted_hits.bam
```

where we have specified the bam file produced from the previous Tophat2 commands. We may want to add a reference annotation to our command. This may seem counter-intuitive since the purpose is to create our own new transcriptome assembly, but doing so allows cufflinks to assign gene names corresponding to known gene names in the annotation file. We can do this with the updated command:

```
cufflinks -g ensembl_mm10.gtf reads_mm10_tophat/accepted_hits.bam
```

Note that we are specifying the same GTF file as before, but cufflinks uses a lower-case "-g", whereas Tophat2 uses an upper-case "-G". We may want to consider is to make the alignment run "quietly", by reducing the number of printed messages with the `-q` option. Another option worth considering is the `-I` option (capital i), which specifies the maximum intron length, or gap-length to consider. Restricting this can occasionally remove artifacts since the default is $300,000$bp.

Measuring Differential Expression with RNA-Seq

When quantifying expression with RNA-Seq, it is important to consider a normalization of the data that best controls for the fact that sequencing depth may vary from experiment to experiment, and gene lengths are highly variable. The simplest of such quantities is RPKM, which stands for Reads Per Kilobase of gene length per Million reads mapped. For a gene g of length L_g, with R_g reads mapping to it, we would compute the expression $RPKM_g$ with

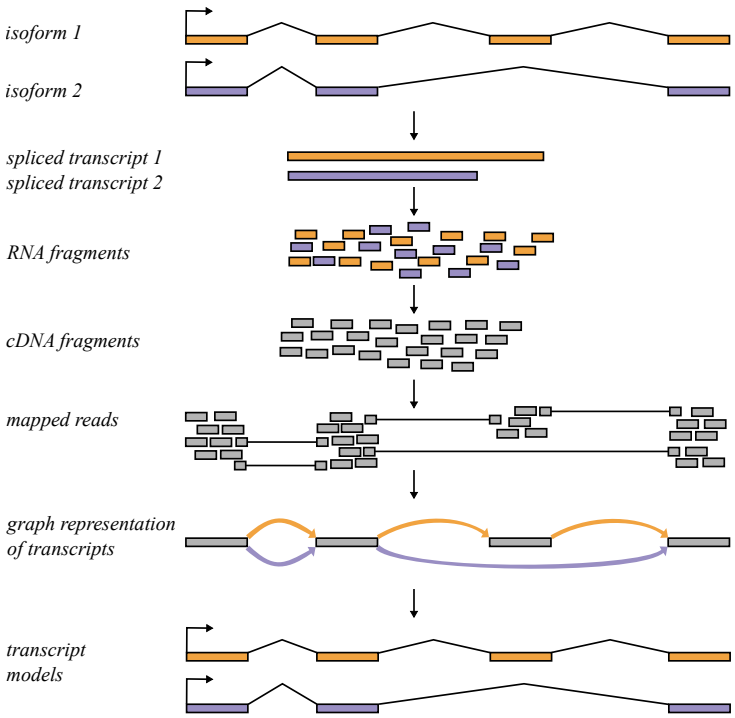

Figure 6.2: A schematic representation of an RNA-Seq pipeline for computation of transcript models

$$RPKM_g = \frac{R_g 10^9}{L_g N} \tag{6.1}$$

A slightly more sophisticated version of this expression is the FPKM, or Fragments Per Kilobase of gene length. Here, the fragment refers to the cDNA fragment from which the read was sequenced. For example with paired-end reads, we have two reads per fragment. If one of the mates was of poor quality or did not map, the reads would underestimate the number of fragments mapping to this location. For example, when aligning with Tophat 2, there is an option "`--no-discordant`", which would only allow reads where both mates properly map the the same chromosome or scaffold. When this option is not used, there are cases where one fragment corresponds to one read (either `R1` or `R2`) or both reads. When the `--no-discordant` option is used, each fragment corresponds to two reads.

The program `cuffdiff` is part of the `cufflinks` software package, and it computes expression values of genes and identifies significantly differentially expressed genes. with default parameters, we can run cuffdiff to compare the expression of two RNA-Seq experiments with the following command (input on one line):

```
cuffdiff -o treatment_vs_control -L treatment,control
ensembl_mm10.gtf treatment_mm10/accepted_hits.bam control_mm10/accepted_hits.bam
```

So we note that in this case it is required that we specify a GTF annotation file to use to use such that the expression of the genes annotated in this file are quantified. Furthermore, we need two bam files, presumably both aligned with Tophat using the same parameters. The `-L` parameter specifies "labels" for the bam file compared, such that the output files will use these in the file header. For each bam file, we could also give sam files, or a comma-separated list of bam files corresponding to individual replicates for that experiment. Finally, the `-o` command specifies the output directory in which the results will be print.

The results of a cuffdiff computation is a set of many files. Among them are differential expression files called `.diff` files. There are files for the genes and for the isoforms for the genes given by `gene_exp.diff` and `isoform_exp.diff` files respectively. These `diff` files contain the information on the genomic location of the transcript, as well as FPKM values for the samples compared. Furthermore, it has the log fold change

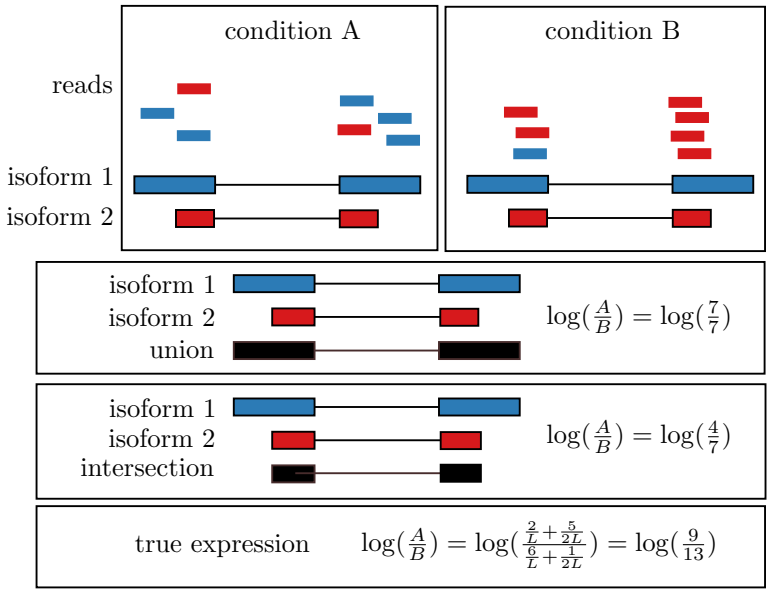

Figure 6.3: A depiction of the intersection and union models, and how they can differ from the actual expression. This diagram is comparable to the Figure 1 from Trapnell *et al.* 2013. In each case, the expression per length is depicted for the two isoforms where each exon of the longer isoform is of length L, and both for the shorter (red) isoform is $L/2$.

of these expression values, as well as output from a statistical test to identify significantly differentially expressed genes.

Intersection Count, Union Count, and the True Expression

When counting reads that overlap a gene model, there are different approaches to quantifying the expression of a gene with multiple transcript isoforms. For example, the intersection count would take the expression of all exons that are common to each isoform in a particular gene. The union count take the perspective of counting the reads that overlap the union of all exons for a particular gene. As pointed out in Trapnell *et al.* 2013, there are issues with both of these models. Figure 6.3 demonstrates a scenario with both the intersection and union count models fail to accurately depict the expression of a gene.

6.2.1 Assembling Transcriptomes without a Reference Genome

There are approaches to assemble a transcriptome and quantify gene/transcript expression without a reference genome. Prominent among them is the suite called "Trinity", which consists of three pieces of software, as the name suggests.

The software components of Trinity are as follows:

1. Inchworm - Cluster reads by common K-mers (subsequences of length K)
2. Chrysalis - Cluster contigs into components consisting of matched K-mers, build de Bruijn Graph from components.
3. Butterfly - Assemble Transcript Models from the de Bruijn Graph

6.3 Transcription Initiation

Transcription Initiation is a remarkable process, because it is very much a needle-in-a-haystack phenomena. Somehow, the cell is able to pinpoint specific single-nucleotide positions out of the genome, and use

these specific positions as the start of a transcript. This first nucleotide that is transcribed is called the Transcription Start Site (TSS).

In many cases, transcription initiation begins with the recognition of the core promoter, which is a collection of binding sites that direct the initiation of transcription. In Eukaryotes, RNA polymerase II does not directly recognize the core promoter sequences. Instead, a collection of proteins called transcription (activation) factors (TAFs) mediate the binding of RNA polymerase and the initiation of transcription. Only after certain transcription factors are attached to the promoter does the RNA polymerase bind to it. The completed assembly of transcription factors and RNA polymerase bind to the promoter, forming a transcription initiation complex.

In prokaryotes, transcription begins with the binding of RNA polymerase to the promoter in DNA. At the start of initiation, the core enzyme is associated with a sigma factor that aids in finding the appropriate -35 and -10 base pairs upstream of promoter sequences. When the sigma factor and RNA polymerase combine, they form a holoenzyme.

6.3.1 Methods of Mapping Transcription Start Sites (TSSs)

There are many methods that have been developed for the mapping of transcription start sites. Most of these methods rely on the biochemistry of the 5' end of transcribed RNA.

Cap Analysis of Gene Expression (CAGE)

CAGE sequencing begins by capping the 5' ends of small transcript fragments, which are then extracted, reverse transcribed to DNA, PCR amplified and sequenced. CAGE reads tend to map at the transcription start site of genes , but in practice do map at and around these locations. The CAGE technique precedes that of high-throughput sequencing and was used before then, but has benefited from the depth associated with new sequencing techniques.

Rapid Amplification of cDNA Ends (RACE)

In RACE, a cDNA copy of the mRNA is produced through reverse transcription, PCR amplification of the cDNA copies. There are different methods for 5' RACE and 3' RACE, resulting in the proper mapping of both the 5' end and the 3' end of genes. 3' RACE uses the poly(A) tail of genes for priming during the reverse transcription. 5' RACE uses a gene-specific primer that recognizes a known region of the gene of interest.

6.4 Promoters

Broadly speaking, the promoter of a gene is the set of genomic DNA sequences that direct the gene's transcription. This can include the core promoter that is the most proximal region around the gene's transcription start site, and can also include other regions of genomic DNA that are involved in transcription initiation.

6.4.1 Core Promoters

The core promoter is a collection of binding sites, capable of directing the initiation of transcription, and located within $+/-40bp$ from the TSS. In eukaryotes, this can include the TATA-box and the Initiator, as well as other motifs that are more organism-specific.

6.4.2 Databases of Promoters/TSSs

There are a number of databases that are devoted to cataloging the locations of the promoters and transcription start sites of genes.

EPD: Eukaryotic Promoter Database

The Eukaryotic Promoter Database (EPD) is a resource of experimentally validated promoter locations for animals, plants, and fungi. This database contains non-redundant entries of promoter locations for Humans, mouse, *D. melanogaster*, zebrafish, *C. elegans*, *Arabidopsis thaliana*, *S. cerevisiae*, and *S. pombe*. The database can be accessed at `http://epd.vital-it.ch/`.

Database of Transcription Start Sites (DBTSS)

The Database of Transcription Start Sites (DBTSS) is a resource of experimentally mapped transcription start sites, through a method called TSS-Seq, and is primarily focused on human and mouse, but contains other species. This database now contains 491 million TSS tag sequences collected from 20 different human tissues and 7 different human cell cultures. The DBTSS can be accessed at `http://dbtss.hgc.jp/`.

The TSS-Seq method ligates the Illumina sequencing adapter to the 5' cap site of the mRNA, performs full-length cDNA creation, and high-throughput sequencing. The uniquely mapping sequencing reads are then mapped to the genome, and clustered into 500bp bins. TSCs overlapping internal exons are then removed, and each cluster is either associated with the most likely RefSeq transcript, or labeled as intergenic.

6.5 Transcription

Transcription is carried out by RNA Polymerase, which makes a RNA copy of the template strand of a gene's DNA sequence. Measurements of the binding of RNA Polymerase (Pol) to genes has revealed that not all genes have the same binding profiles over their promoter regions and gene bodies.

6.5.1 Measuring RNA Polymerase binding: Pol II ChIP-Seq

Pol II ChIP-Seq provides a signal the binding of Pol II to the genomic DNA. It measures the recruitment of Polymerase to the genome, but the Ser5 of the Pol II complex needs to be phosphorylated for transcription to begin. Therefore, many of these protocols use antibodies that specifically recognize the Ser5-Phosphorylated form of Pol II as well as other forms.

Briefly, chromatin samples are crosslinked and sonicated into fragments. Chromatin is immunoprecipitated with a RNA Pol II antibody, or cocktail of antibodies that recognize different forms of Pol II. DNA fragments are then isolated and sequenced with HTS. These reads are then mapped to the genome, and the number of reads that map to a particular genomic region is indicative of the time that Pol II spends binding to that region.

6.5.2 RNA Polymerase II Stalling

One of the phenomena observed using Pol II ChIP-Seq is RNA Polymerase II Stalling (Pol II Stalling). Pol II Stalling is associated with a greater proportion of Pol II ChIP-Seq reads at and around the TSS of genes compared to the gene body. For a Pol II signal given by $S(i)$ for position i of the genome, The state of being stalled is defined by a large stalling index:

$$SI = \frac{max_{TSS}\{S(i)\}}{median_{gene}\{S(i)\}}$$

where $max_{TSS}\{S(i)\}$ indicates the maximum signal value $S(i)$ within some distance of the TSS of the gene, and $median_{gene}\{S(i)\}$ indicates the median signal value within the gene body.

6.6 Elongation

Elongation is characterized by the release of Pol II from the promoter, and the production of RNA transcripts. In mammalian genes, the Pol II elongation rates are about 0.5kb per minute in the first few kilobases, and increases to 2-5kb per minute after 15kb [25].

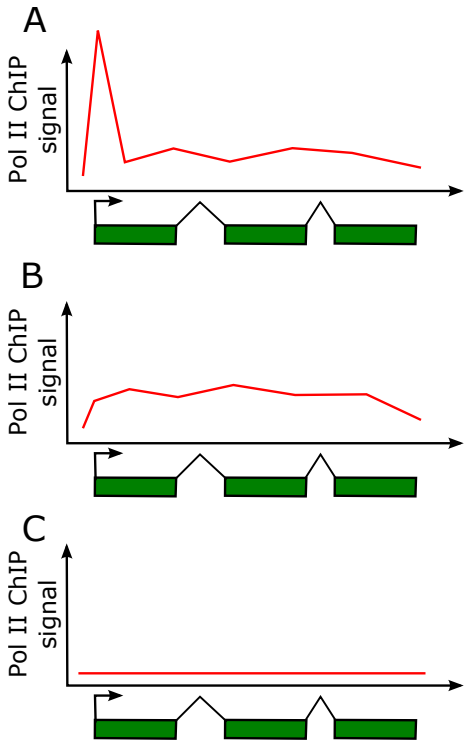

Figure 6.4: The Pol II signal over a gene can indicate its transcriptional status. A. Genes with a large proportion of Pol II binding at the TSS are "Stalled". B. Genes with roughly uniformly high Pol II binding for the promoter compared to the gene body are called active. C. Genes with little or no Pol II binding are categorized as "No Pol II".

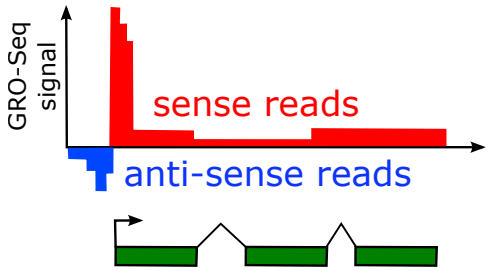

Figure 6.5: GRO-Seq can identify cases of divergent transcription, where nascent transcription is observed both sense and antisense to a genes TSS.

6.6.1 Measuring Nascent Transcription: GRO-Seq

While Pol II ChIP-Seq measures DNA-associated Pol II, Global Run-On Sequencing (GRO-Seq) measures transcriptionally engaged Pol II, by detecting nascent RNA transcripts. GRO-seq involves a nuclear run-on experiment on a genome-wide scale, where engaged Pol II incorporates bromo-tagged nucleotides, and is followed by RNA isolation and next-generation sequencing.

6.6.2 Divergent Transcription

One of the phenomena observed using Global Run-On sequencing (GRO-Seq) is that of divergent transcription. Divergent transcription is characterized by nascent transcripts associated with both the sense and anti-sense strand of the gene.

6.7 Gene Expression

Gene expression studies measure the expression levels of genes by attempting to quantify the number of mRNA transcripts per gene. There are a number of ways to compute gene expression, with varying accuracy and different pros and cons.

6.7.1 Microarrays

Microarrays allow for high-throughput measurement of gene expression through the use of hybridization probes. The probes are DNA sequences that are complementary to the transcripts that you would like to measure (or more precisely the cDNA created from the transcripts). Because of the requirement of probes to detect the gene expression, one or more probes for each gene needs to be designed. This puts a constraint on the genes that one can detect. Also, different hybridization energies can complicate the analysis in some cases, as this would need to be controlled for.

6.8 Small RNA-Seq

When sequencing small RNAs, there are other considerations when doing RNA-Seq. Sequencing of size-selected small RNA samples using high-throughput sequencing can be called Small RNA-Seq. Such a protocol can be used to sequence RNA species such as microRNAs and piRNAs whose endogenous mature nucleotide sequences can be shorter than the read length used to sequence them. In such cases, the 3' adapter sequence needs to be removed before aligning these sequences. For example, the program `cutadapt` can be used to trim adapter sequences from an input FASTQ file.

```
cutadapt -a ATCTCGTATGCCGTCTTCTGCTTG reads.fastq > reads_trimmed.fastq
```

where we are trimming off the Illumina TruSeq Small RNA 3' adapter sequence.

Chapter 7

Noncoding RNAs

It was once believed that most genes encode proteins and most transcripts are messenger RNAs (mRNAs). According to the most recent studies on the matter, the number of genes that do not encode proteins outnumbers those that do. Unlike mRNAs, which are translated into proteins by the ribosome, these noncoding RNAs (ncRNAs) act functionally as an RNA transcript. NcRNAs that are greater than 200nt are called long noncoding RNAs (lncRNAs), and those that are less than 50nt can be called small RNAs (smRNAs). The remaining intermediate size range from 50-199nt can be called short RNAs, although these exact length thresholds are a bit arbitrary.

Many ncRNAs operate by forming a structure, consisting of basepairs between nucleotides (formed by hydrogen bonds), as well as regions of unpaired nucleotides. An RNA structure can be viewed at many different levels, with the "primary structure" consisting of the sequence itself. The secondary structure consists of a mapping of which nucleotides are paired, and can be represented on a 2D surface. The tertiary structure consists of the three dimensional configuration of the molecule. The quaternary structure comprises the inter-molecular interactions that these molecules can participate in.

While the primary structure of the RNA can be described by a sequence r of the characters $\{A, C, G, U\}$, such that $r[i]$ corresponds the the nucleotide at position i of the molecule the sequence represents. The secondary structure can be represented by a set of ordered pairs S such that each pair of positions $i < j$ that form a base pair constitutes an ordered pair $(i, j) \in S$, and the corresponding nucleotides $r[i]$ and $r[j]$ are complementary.

7.1 Duplexes: Nearest Neighbor Model

Consider the complementary RNA sequences $A =$ `ACGCAUU` and $B =$ `AAUGCGU`. The process by which the these single stranded nucleotide sequences A an B form a base-paired (heterodimeric) RNA duplex AB is given by the following chemical reaction

$$A + B \rightleftharpoons AB$$

This is under the assumption that there are no structures formed by A and B on their own that need to unfold for the reaction to proceed forward, and that the individual molecules A and B don't form homodimers on their own.

In many applications, this equation is depected the other way, and the process is described as a "melting" process whereby the basepairs become unpaired:

$$AB \rightleftharpoons A + B$$

The energy required to drive such a reaction is set by the equilibrium constant for this reaction

$$K = \frac{[A][B]}{[AB]}$$

And the free energy required to melt the duplex is given by

Sequence	ΔH (kcal/mol)	ΔS (kcal/K mol)	ΔG (kcal/mol)
AA/UU	-6.82	-19.0	-0.93
AU/AU	-9.38	-26.7	-1.10
UA/UA	-7.69	-20.5	-1.33
CA/UG	-10.44	-26.9	-2.11
GU/AC	-11.4	-29.5	-2.24
CU/AG	-10.48	-27.1	-2.08
GA/UC	-12.44	-32.5	-2.35
CG/CG	-10.64	-26.7	-2.36
GC/GC	-14.88	-36.9	-3.42
GG/CC	-13.39	-32.7	-3.26
init.	3.61	-1.5	4.09
symm	0	-1.4	0.43
per term-AU	3.72	10.5	0.45

Table 7.1: Base pair energies for the Nearest Neighbor model. The init terms assume that duplex has at least one G·C basepair. The units for Entropy are "entropy units" measured in 4.184J/Kmol

$$\Delta G = -RT\ln(K) = -RT\ln\frac{[A][B]}{[AB]}$$

where $R = 1.985877510^{-3}kcal/molK$ is the ideal gas constant in kcal/molK.The temperature at which there are equal proportions of paired and unpaired RNA strands is defined as the melting temperature, which is computed by:

$$T_m = -\frac{\Delta G}{R\ln([AB]_0/2)}$$

7.2 Nearest Neighbor Model

In our example above, the RNA duplex with sequences $A =$ `ACGCAUU` and $B =$ `AAUGCGU` forms a duplex. Since this system is completely paired, we can describe it as a sequence of base pairs b_i, corresponding to the base pair at position i. In this example, using 1-based coordinates, we have $b_1 = A \cdot U$. The nearest neighbor model describes the basepair energy between a duplex as the sum of dimer interactions, plus initialization terms on the ends. The full expression for the duplex energy is then

$$\Delta G = \Delta G_{init}(b_1) + \Delta G_{init}(b_n) + \sum_{i=1}^{n-1} \Delta G(b_i, b_{i+1})$$

where $\Delta G(b_i, b_{i+1})$ is the free energy associated with the dimer pair formed by basepair b_i at position i, and basepair b_{i+1} at position $i+1$ of the duplex. Each such $\Delta G(b_i, b_{i+1})$ corresponds to the stabilizing energy of the hydrogen bonds, but also the stacking energy between pairs of neighboring nucleotides. These values are presented in Table 7.1. This table can be understood such that the term X_1X_2/Y_1Y_2 means that the first base pair is of $X1 \cdot Y_1$ and the second basepair is $X_2 \cdot Y_2$, which is to say the dimer X_1X_2 (5' to 3') pairs to the dimer Y_2Y_1 (5' to 3').

7.3 Destabilizing energies

This nearest neighbor model so far only considers the stabilizing energy (negative terms) of a base pair stem, and does not consider the destabilizing energy of structures such as hairpin loops that contributed positive, destabilizing energy to a structure.

length (nt)	C·G-closed loop	A·U-closed loop	bulge loop
1	999	999	2.8
2	999	999	3.9
3	8.4	8.0	4.5
4	5.9	7.5	5.0
5	4.1	6.9	5.2
6	4.3	6.4	5.3
7	4.5	6.6	5.5
8	4.6	6.8	5.6
9	4.8	6.9	5.7
10	4.9	7.0	5.8
12	5.0	7.1	5.9
14	5.2	7.3	6.1
16	5.3	7.4	6.2
18	5.4	7.4	6.3
20	5.5	7.6	6.4
25	5.7	7.7	6.5
30	5.9	7.9	6.7

Table 7.2: The destabilizing energy of hairpin loops with closing C·G basepair, with A·U basepair, and for a bulge loop in kcal/mol.

The destabilizing energy of a hairpin can be summarized in table 7.2. For a loop of length n, the destabilzing energy caused by the loop can be approximated by:

$$\Delta G_{loop}(n) = 1.75 \times RT \times \ln(n),$$

but this equation is only valid for $n \geq 10$.

7.4 RNA Secondary Structure Prediction

The Nussinov Algorithm predicts the number of basepairs, and ultimately the structure for an RNA sequence using dynamic programming. This is done by understanding that whether or not a pair of nucleotides i, j will pair depends on the intervening nucleotides from $i+1$ to $j-1$.

Let's consider an RNA sequence r with nucleotides $r[i]$. Consider the possible base pair formed between positions i and j. This will of course depend on whether $r[i]$ and $r[j]$ are complementary or wobble pairs capable of forming hydrogen bonds.

Let's build a scoring matrix B such that the terms of this matrix B_{ij} correspond to the number of basepairs between these two positions. There are four possibilities to consider when evaluating the structure between positions i and j. These four possibilities are depicted in Figure 7.1. The recursion relation that describes this process is given by the following relation:

$$B_{i,j} = max \begin{cases} B_{i+1,j-1} + 1 & i \text{ and } j \text{ are paired} \\ B_{i+1,j} & i \text{ unpaired, use structure from } i+1 \text{ to } j \\ B_{i,j-1} & j \text{ unpaired, use structure from } i \text{ to } j-1 \\ max_{i<k<j}\{B_{i,k} + B_{k+1,j}\} & \text{two sub-structures} \end{cases} \tag{7.1}$$

The first three options are probably pretty clear. Either it adds a basepair at i, j, or it doesn't. If it adds a basepair, then increment +1 to the previous value $B_{i+1,j-1}$, that corresponds to the best structure prediction for the intervening positions. If it doesn't add a pair, then simply apply a previously computed number of basepairs to B_{ij}. The fourth term is the most complex. In this condition, there is an intermediate nucleotide k, such that the structures produced from i to k and from $k+1$ to j are optimal. After going

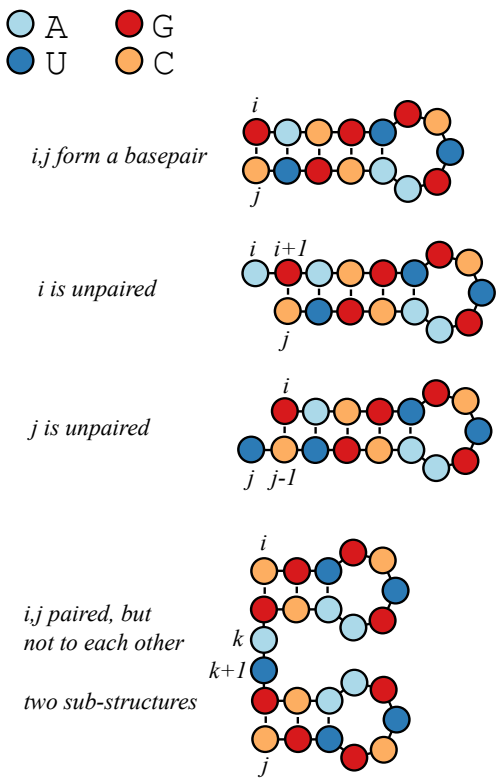

Figure 7.1: The recursion relation in the Nussinov algorithm can be broken up into four possibilities.

through all positions, the optimal number of basepairs for a sequence of length n corresponds to the value of $B_{1,n}$

Such a method only considers the base pairs, and does not consider that not all basepairs contribute the same energy, as seen in the tables above. Furthermore, it does not take into account the destabilizing energy of the loops, bulges, and other unpaired structures. For example, Figure 7.2 shows some of the common forms of destabilizing energies in RNA structures. Fortunately, there are other algorithms that do compute the minimum free energy including the contribution of these types of structures, and have already been implemented in relatively easy to use software.

7.4.1 RNAfold

`RNAfold` is software that comes as part of the Vienna RNA package. To compute a structure for a sequence, simply input the sequence data as a FASTA file using `cat`. The GNU/Linux program `cat` simply prints the file to the screen, but upon piping it into a file will run the program. For example, consider the microRNA hairpin sequence for miR-1 in humans:

```
$ cat mir-1.fasta | RNAfold --noLP -p
>hsa-mir-1-1 MI0000651
UGGGAAACAUACUUCUUUAUAUGCCCAUAUGGACCUGCUAAGCUAUGGAAUGUAAAGAAGUAUGUAUCUCA
(((((.((((((((((((((((..(((((((.....)))...))))).))))))))))))))))).))))) (-29.10)
{((((.((((((((((((((((..(((((((.....)}...))))).))))))))))))))))).))))} [-29.84]
(((((.((((((((((((((((..(((((((.....)))...))))).))))))))))))))))).))))) {-29.10 d=2.64}
 frequency of mfe structure in ensemble 0.299677; ensemble diversity 3.99
```

The result is a prediction of the minimum free energy structure for the molecule (in dot-bracket notation), and the minimum free energy of the structure (in kcal/mol). The dot-bracket notation can be understood as containing matching parentheses that correspond to basepairs. Each left parenthesis has a corresponding right parenthesis, and their positions match which nucleotides are predicted to be paired. Dots correspond to

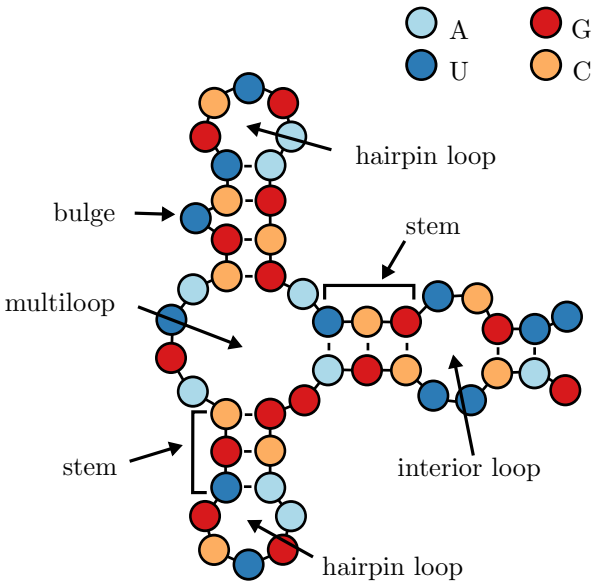

Figure 7.2: RNA structures can be composed of many different structural features.

unpaired nucleotides. In addition, by default, two postscript image files are created. The first is a secondary structure image ending in `_ss.ps` and the second is a dotplot image ending in `_dp.ps`. The resulting image for this sequence is shown in Figure 7.3

The secondary structure image is a representation of how the structure of the RNA would look in a 2D plane. In a real 3D tertiary structure, there would be helical turns in the stem region of the structure. The dotplot figure shows a matrix where below the main diagonal demonstrates the base pairs present in the minimum free energy structure. Each term of the matrix corresponds to a pair i, j of positions along the sequence. If the two nucleotides at positions i and j are paired, then that cell of the matrix is filled. Above the main diagonal shows the probability of each basepair. You will notice some faint dots around the main larger ones, corresponding to basepairs that could form in suboptimal configurations.

Next, let's consider computing the structure of a tRNA. Suppose that the FASTA file `trna.fasta` contains a tRNA for the yeast species *Saccharomyces cerevisiae*. We can compute the structure as before with the command:

```
$ cat trna.fasta | RNAfold --noLP -p
>Saccharomyces_cerevisiae_chrXVI.trna12-PheGAA (622631-622540)  Phe (GAA) 92 bp  Sc: 69.79
GCGGAUUUAGCUCAGUUGGGAGAGCGCCAGACUGAAGAAAAAACUUCGGUCAAGUUAUCUGGAGGUCCUGUGUUCGAUCCACAGAAUUCGCA
((((((...((((........))))....(((((((((......)))))))))......((((..((((........))))..)))))))))). (-30.40)
((((((,..((((........)))),{..(((((((((......)))))))))..},..{((((..((((........))))..)))))))))). [-31.39]
((((((...((((........))))....(((((((((......)))))))))......((((..((((........))))..)))))))))). {-30.40 d=3.52}
 frequency of mfe structure in ensemble 0.199156; ensemble diversity 5.29
```

The result is something that looks kind of like a cloverleaf structure. An important fact about tRNAs is that they are chemically modified in the cell, which results in a different structure than one might predict. In the end, the specific structure formed by a tRNA probably looks different than the predictions here due to these modifications. The images produced by `RNAfold` are depicted in Figure 7.4.

7.5 RNA/DNA Hybrids

RNA/DNA hybrids are structures formed by a "hybrid duplex" consisting of one strand of RNA and one strand of DNA. However, in order for the RNA/DNA hybrid to be made, the DNA duplex needs to be opened. Therefore, the energy parameters of such an interaction can be computed using the difference between the DNA/DNA duplex energy and the RNA/DNA hybrid energy. The $\Delta\Delta G$ can be computed from the difference of these two values. Therefore, the formula is given by:

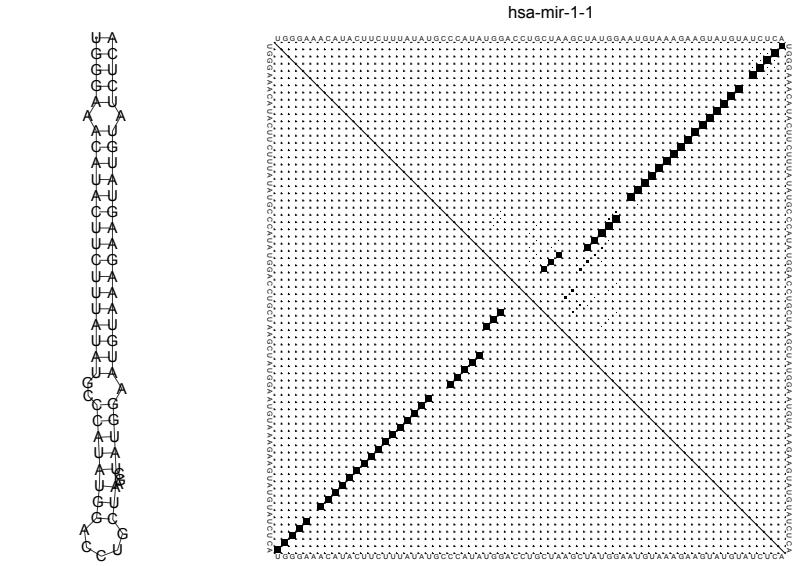

Figure 7.3: The structure and dot plot for a microRNA

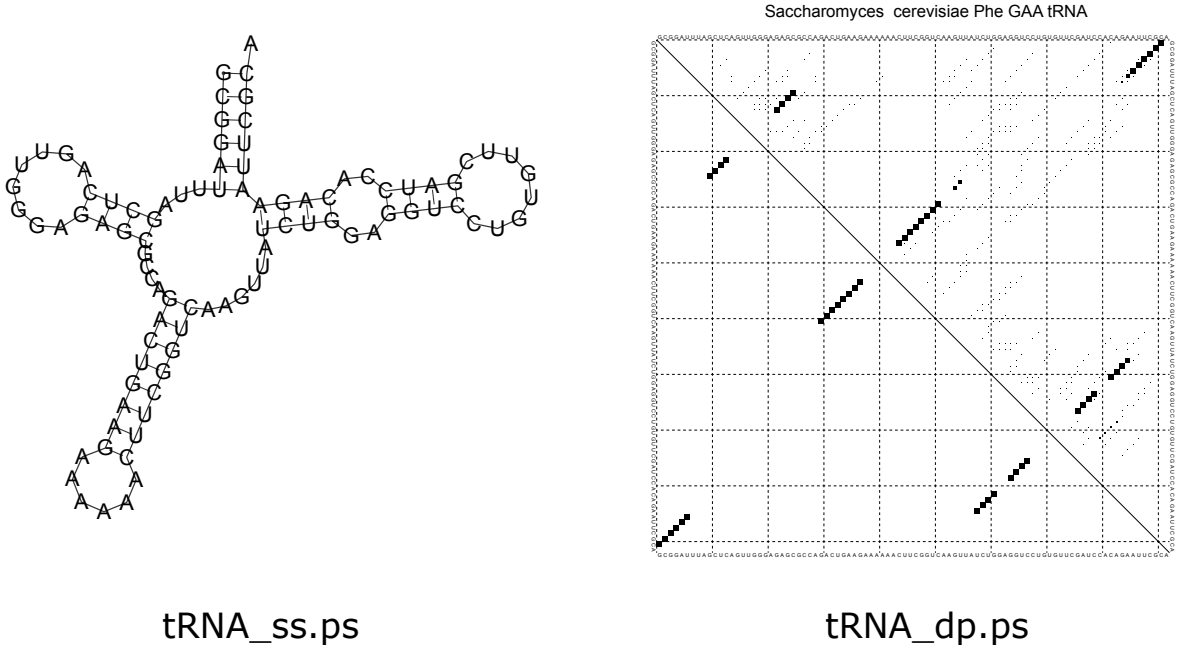

Figure 7.4: The structure and dotplot for a tRNA

DNA/DNA duplex

first → second ↓	G	C	A	T
G	-1.84	-2.17	-1.28	-1.45
C	-2.24	-1.84	-1.44	-1.30
A	-1.3	-1.45	-1.00	-0.58
T/U	-1.44	-1.28	-0.88	-1.00

RNA/DNA hybrid

G	C	A	U
-2.2	-1.2	-1.4	-1.0
-2.4	-1.5	-1.6	-0.8
-1.4	-1.0	-0.4	-0.3
-1.5	-0.9	-1.0	-0.2

RNA/DNA - DNA/DNA

first → second ↓	G	C	A	T/U
G	-0.36	0.97	-0.12	0.45
C	-0.16	0.34	-0.16	0.5
A	-0.1	0.45	0.6	0.28
T/U	-0.06	0.38	-0.12	0.8

Table 7.3: Parameters for calculating RNA/DNA hybrids for exact matches.

$$\Delta\Delta G = \Delta G_{hybrid} - \Delta G_{DNAduplex}$$

The values for any exact matching dimers can be retrieved from the following table, which contains RNA/DNA parameters, DNA/DNA values, and their difference.

7.6 Triplexes

Triple-helix (triplex) structures between a double-stranded nucleic acid sequence and a single-stranded nucleic acid sequence can form for specific triplets of nucleotides. For example, triplexes can form between a single-stranded RNA sequence and double-stranded DNA. The energy parameters of these interactions are not known.

7.7 Quantifying coding potential

In order to identify noncoding RNAs, perhaps it is easier to characterize what is coding, and label the remaining as noncoding. Many approaches have been developed to quantify a transcripts potential to encode a protein ranging from comparative approaches, to machine learning, to experimental methods.

7.7.1 Homology

The first approach is to use homology, such as by BLAST or using protein domains on predicted ORFs. If there is significant homology between the translation of an ORF and a known protein, then there is a good chance that the gene in question is also protein coding. Similarly, if an ORF translates to a sequence that has a known protein domain, then it is likely protein coding.

7.7.2 Evolutionary Models

Evolutionary methods for identifying coding potential rely on the fact that protein coding genes have well defined sequences of codons, which constitute an open reading frame. Open reading frames (ORFs) are defined by a start codon positioned at a multiple of three nucleotides away from a stop codon. When considered comparatively across different species in a multiple alignment, one rarely sees nucleotide insertions or deletions that cause a "frame shift". Frame shifts occur when the phase of the codons changes, giving rise to a transcript that encodes a completely different amino acid sequence, if at all. Similarly, multiple sequence alignments of protein coding transcripts are depleted for mutations that cause a non-synonymous change, or mutations that alter which amino acid a codon encodes. The software PhyloCSF does exactly

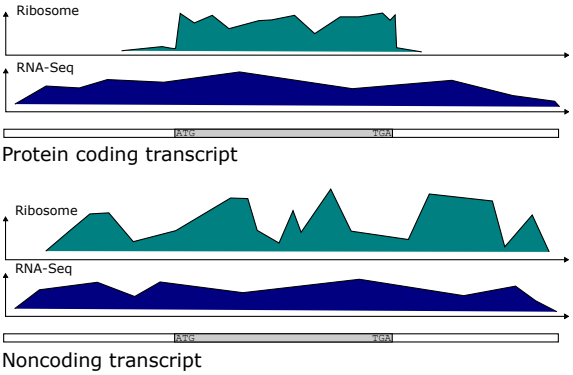

Figure 7.5: While protein coding genes exhibit a sharp decrease in ribosome binding after the stop codon, binding to noncoding RNAs is more stochastic.

this, and quantifies the likelihood of a given transcripts being coding or noncoding based on its multiple alignment [26].

7.7.3 Machine Learning Approaches

Machine learning approaches define a set of "features" that describe the phenomena that the program seeks to learn. For identifying protein coding genes, the features describe aspects of the transcript being evaluated. For example, the program `CPAT` (Coding Potential Assessment Tool) quantifies the frequencies of K-mers in the transcript, as well as the length of the transcript, and is able to distinguish protein coding genes from noncoding fairly well [27].

7.7.4 Experimental Validation

Ribosomal Binding Profiles

Ribosomal profiling quantifies the binding of the Ribosome to transcripts at each position. It has been determined that protein-coding genes have a distinct ribosomal profile over the open reading frame, characterized by a sharp drop-off in binding immediately after the stop codon. Noncoding RNAs have a more stochastic binding profile around the stop codon [28]. Figure 7.5 depicts the kind of data produced in these experiments. To quantify the potential for coding, the authors developed a score called a Ribosomal Release Score (RRS). The equation for the RRS is

$$RRS = \frac{(C_{CDS}/C_{3'UTR})_{Ribosome}}{(C_{CDS}/C_{3'UTR})_{RNA-Seq}} \tag{7.2}$$

For protein-coding genes, we expect this ribosome ratio to be high, since the ribosome read count C_{CDS} should be much greater than that of $C_{3'UTR}$ for such genes.

Mass Spectrometry

Some approaches have used mass spectrometry to detect small peptides, and validate that a particular open reading frame was real. In these cases, small peptides can be detected that were predicted to be translated from a small ORF [29].

Chapter 8

Proteins

Many transcripts encode Proteins, which are sequences of amino acids that fold into specific structures responsible for their functions. To a great extent, the sequence of the protein determines its structure, which in turn determines its function. Therefore, identifying similar protein sequences can suggest structures and functions. The field of protein bioinformatics is certainly extensive, and we will only be able to cover a small portion to introduce some concepts that may be useful.

8.1 Identifying ORFs

To go from mRNA to protein, we need an open reading frame (ORF) defined by a start codon in frame with a stop codon. Typical mRNAs have many possible ORFs, with other signals, such as the Kozak sequence, determining which ORF is used. In most cases, the longest ORF is the coding DNA sequence (CDS), or the region that is actually translated. In general it is important to distinguish ORFs, which are quite common, from the CDS.

8.2 Inferring Protein Function

Computational methods can be used to infer protein function when a new protein is identified, but even if the inference is very strong, experiments still need to be done to validate the prediction. That said, there are many ways in which a hypothesis about protein function can be inferred:

1. Sequence Homology
2. Sequence Motifs: "Domains"
3. Structure-based Function Prediction
4. Genomic Co-location (gene clusters)
5. Expression data (Co-expression)

8.2.1 Protein Evolution and Homology

The first method that one might use to infer protein function is by sequence homology. Frequently, the more fundamental molecular functions of proteins are very well conserved. There are a number of mechanisms of protein evolution that one must consider when evaluating protein homology.

In DNA replication, there are often errors that cause major changes in proteins, which can be highly disruptive to the function of the protein, but perhaps in some cases may confer new function. First, is the event of gene fusion, where two genes may be merged into one gene, combining different functional regions from the two ancestral genes. For example, a gene encoding a protein with a DNA-binding domain could fuse with a gene encoding a protein with a protein-binding domain, producing a new gene encoding a protein

with both domains. In many cases, a gene fusion event is caused by a chromosomal translocation event, but such translocations can also cause other mutations in genes, such as loss of a portion of a gene. In addition, regions of chromosomes can be inverted during replication, resulting in gene fusions or deletions of portions of genes. Along the same lines, a chromosomal deletion, where a portion of a chromosome is deleted during DNA replication, can also lead to gene fusion events or loss of a portion of a gene.

Gene duplication events can happen as part of a chromosomal duplication, or as part of a local duplication. When this happens, multiple copies of a gene can be created. The two copies of the gene could be beneficial in some cases, or harmful in others. In these cases, one of the copies can be possibly silenced. When both copies remain, one of the copies can undergo less selective pressure, because the bulk of the work is carried out by one of them. In this case, the other copy can accumulate mutations over time, which could result in new function or a specialized function. This process where one copy accumulates mutations and ultimately carries out a specialized role is often called sub-functionalization of the protein.

1. Gene Fusion

   Translocation

   Interstitial Deletion

   Chromosomal Inversion

2. Gene duplication and subfunctionalization

   Chromosomal duplication

   Local duplication events

8.3 Similarity Matrices

Typically, when aligning two sequences, the assumption is that the two sequences have a common evolutionary origin, and the association between single characters in the alignment correspond to the evolutionary history of those characters. In order to optimize such an alignment, we need to define a scoring system used to compare two characters.

Let's define some probabilistic models of sequence alignments so that we can describe the scoring system with such a formalism. Much of what we will describe here was developed for protein sequences, but the formalism could be used more generally. First, let's formulate the probability of the two sequences given a random model. That is, the two sequences are just two random strings of characters that have been independently generated from a generative probabilistic model.

To compute this, let's assume an ungapped alignment of the two sequences x and y. Assuming each of the characters $x[i]$ and $y[i]$ at each position i is just randomly selected according to their frequency in the database, then the simplest model would describe the probability as the product of these single character frequencies.

$$P(x, y|Random) = \prod_{i=1}^{n} p_{x[i]} p_{y[i]}$$

Alternatively, if the sequences have a common ancestor, we could also define a probability $q_{a,b}$ as describing the frequency of occurrence of substitutions in the database between two characters a and b. Therefore, we have the expression

$$P(x, y|Ancestor) = \prod_{i=1}^{n} q_{x[i],y[i]}$$

where the total probability is described as the product of the frequencies of each substitution in our database. At this point, it is important to note that the probability $q_{a,b}$ can be expressed as a product of the frequency of occurrence of a times the conditional probability of a character mutating to b given that it was a. Therefore,

$$q_{a,b} = P(a)P(b|a) = P(a)P(a \to b)$$

Furthermore, we can describe the system as (time-reversible) if and only if

$$q_{a,b} = P(a)P(b|a) = P(b)P(a|b) = q_{b,a}$$

Under such a time-reversible model, the observed frequency of a mutating to b is equal to the observed frequency of b mutating to a. This is a common assumption, particularly because we can't observe, and can only infer, which character is ancestral. We will see below that we can compute the most likely or most parsimonious ancestral character, but the databases that were originally used to build these scoring matrices just used alignments; hence, we can only observe substitutions.

If we combine our two probabilistic models, we can compute a likelihood ratio

$$\frac{P(x,y|Ancestor)}{P(x,y|Random)} = \frac{\prod_{i=1}^{n} q_{x[i],y[i]}}{\prod_{i=1}^{n} p_{x[i]}p_{y[i]}}$$

and by taking a logarithm, we can define a score as a log-odds ratio, which is more convenient for our purpose because it turns the product into a sum

$$S = \log\left(\frac{P(x,y|Ancestor)}{P(x,y|Random)}\right) = \log\left(\frac{\prod_{i=1}^{n} q_{x[i],y[i]}}{\prod_{i=1}^{n} p_{x[i]}p_{y[i]}}\right) = \sum_{i=1}^{n} \log\left(\frac{q_{x[i],y[i]}}{p_{x[i]}p_{y[i]}}\right)$$

The last terms give us our similarity matrix terms $S_{a,b}$ defined as

$$S_{a,b} = \log\left(\frac{q_{a,b}}{p_a p_b}\right)$$

Therefore, we can use the terms of the scoring matrix $S_{a,b}$ to score our alignments. Most of the similarity matrices that have been defined take this form or very similar. Let's examine two such scoring matrices.

8.3.1 PAM Matrix

A point mutation is a single nucleotide or single amino acid mutation, typically surrounded by conserved nucleotides or amino acids. The PAM matrix seeks to compute a substitution matrix using statistics from point mutations in curated protein alignments.

A Point Accepted Mutation Matrix (PAM Matrix), was the first systematically defined matrix for scoring the similarity of peptide sequences [21]. We've just shown that to compare a protein sequence, we need a scoring matrix that compares the similarity of pairs of amino acids. The calculation of such terms involves examining databases of curated protein alignments that contain a certain number of "point accepted mutations", or mutations that involve a single amino acid change. The construction of the PAM matrix begins by defining a mutation matrix M with terms M_{ab} that gives the probability of mutating from a to b. That is,

$$M_{ab} = P(a \to b) = P(b|a).$$

If our database has n_a occurrences of amino acid a, then we define our normalized frequency p_a of that amino acid as

$$p_a = \frac{n_a}{N}$$

where N is the total number of amino acids in the database. Because $\sum_a n_a = N$, the probabilities p_a are normalized. One of the key concepts about the PAM matrix is time. We can't measure how much time has transpired between mutations in the sequences of our database. Therefore, we have to talk about how many substitutions have taken place as a percentage. The amount of time described by a PAM matrix is the amount of time that gives rise to about 1% amino acid change, or 1 substitution per 100 amino acids. Therefore, the probability of no mutation is 0.99. The probability of no mutation is the sum of the products of the probability of each amino acid times the probability that amino acid not mutating, summed over all

amino acids. The probability of a given amino acid a to not mutate is given by the diagonal terms of the mutation matrix M_{aa}, therefore

$$\sum_{a=1}^{20} p_a M_{aa} = 0.99$$

Therefore, this equation scales the mutation rates, and gives us restrictions on the protein alignments that we can consider in our database. We can get more distant evolutionary relationships by defining a series of mutation matrices, defined as the product of the mutation matrix M with itself. To see this, consider the equation:

$$\sum_b M_{ab} M_{bc} = \sum_b P(a \to b) P(b \to c)$$

So the result of this matrix product gives the probability of mutating from a to c in twice the amount of time defined by the matrix M, and considering all intermediate amino acids b. It computes this by first considering the intermediate amino acid b with the term $P(a \to b)$, and then to c with the term $P(b \to c)$, and sums over transitions between all possible intermediate amino acids. If the mutation matrix M_{ab} for the first PAM similarity matrix PAM_1 is M_1, then we can define the nth mutation matrix corresponding to PAM_n as

$$M_n = (M_1)^n$$

The actual PAM matrix that is used in scoring is defined as

$$PAM_1(a, b) = \log\left(\frac{p_a M_{a,b}}{p_a p_b}\right) = \log\left(\frac{M_{a,b}}{p_b}\right)$$

8.3.2 BLOSUM Matrix

The BLOSUM Matrix is a series of matrices that are have been built from the Blocks Database, which is still available at `http://blocks.fhcrc.org` [22]. The blocks database is a database of ungapped protein alignments, such that the proteins aligned satisfy some minimal level of conservation. The BLOSOM matrices are built from them, much like the PAM matrices, but they are defined differently. Consider a single column of one of these Block alignments

```
...L...
...L...
...L...
...I...
...I...
...V...
...V...
```

In this example, there are 3`L`s, 2 `I`s, and 2 `V`s. The next step is to build a frequency matrix, of the number of pairs in the Block alignment that contains the particular pair i and j in a column given by f_{ij}. When the amino acid is the same, for example f_{LL}, this frequency is the number of pairs that can be selected from n_L items, or $\binom{n_L}{2}$, where n_L is the number of occurrences of the amino acid `L`. When the amino acids are different, such as f_{IL}, the value is computed as the product of the number of occurrences of each amino acid, such as $f_{IL} = n_L \times n_I = 3 \times 2 = 6$. To avoid double counting, we can only consider cases when $i \geq j$. These frequencies are normalized to make a q matrix, with terms defined as

$$q_{ij} = \frac{f_{ij}}{\sum_{i=1}^{20} \sum_{j=1}^{i} f_{ij}}$$

The second sum in the denominator just goes up to i to count half of the symmetric matrix f_{ij}, and to avoid double-counting pairs. We can get the normalized frequencies of each animo acid from these quantities by

$$p_i = q_{ii} + \frac{1}{2}\sum_{j \neq i} q_{ij}$$

where the 1/2 term is because there is a probability of 1/2 that a random amino acid selected from the pairs corresponding to the q_{ij} pairs is i. For this equation, imagine selecting a pair of amino acids from the database, then selecting one of the two amino acids from that pair at random. The result should be the amino acid frequency from the database p_i. If the amino acids are the same, and both i then there is a 100% chance when you select one of them it will be i (the first term in the sum). If you select a pair that is different, then the probability is 1/2 when you select one that it will be i (the second term in the sum). This equation for p_i should, however, be equivalent to computing the normalized frequency of occurrence in the alignment database.

Next we want to compute the probability e_{ij} of selecting the amino acids i and j by chance.

$$e_{ij} = \begin{cases} p_i p_j, & \text{if } i = j \\ p_i p_j + p_j p_i = 2 p_i p_j, & \text{if } i \neq j \end{cases}$$

To understand this equation, imagine selecting a pair of amino acids at random from the database. If they are the same, then the probability of that happening is just $p_i p_i = p_i^2$ (the first condition). If they are different, then there are two ways that could be selected: first i then j, or first j then i (the second condition).

So the final values are rounded from the following log-likelihood equation:

$$S_{i,j} = 2 \times \log_2 \left(\frac{q_{i,j}}{e_{i,j}} \right)$$

In practice, the values for the BLOSUM matrices are rounded to integers because when they were created computer memory and computation was expensive, and this required less space than storing decimals and performing floating point arithmetic.

8.3.3 Karlin and Altschul Generalization

Karlin and Altschul formulated a generalized scoring matrix that is similar to both PAM matrices and BLOSUM matrices, defined as

$$S_{i,j} = \left(\frac{1}{\lambda} \right) \ln \left(\frac{q_{i,j}}{p_i p_j} \right)$$

where λ is a positive parameter that scales the matrix [23].

8.3.4 Biopython and Substitution Matrices

We can access values of a given substitution matrix using Biopython and the module `Bio.SubsMat`. For example, the most commonly used substitution matrix and the default for NCBI protein BLAST is BLOSUM62. We can print the values of the matrix or access a particular term by the following

```
>>> from Bio.SubsMat import MatrixInfo
>>> S = MatrixInfo.blosum62
>>> S['Q','Q']
5
>>> S['W','Q']
-2
```

If you play around with this matrix (actually it is a dictionary that takes a pair of characters as keys) you will realize that some pairs are stored, but others are not and return errors when one tries to access them. For example, `S['W','Q']` is stored, but not `S['W','Q']`. To get around this, an additional function can be created:

```
>>> def getScore(S,a,b):
...     if (a,b) not in S:
...             return S[b,a]
...     return S[a,b]
...
>>> getScore(S,'Q','W')
-2
```

8.3.5 Protein Domains

A protein domain is a conserved part of the protein that has a distinct structure and function. Often domains occur as highly conserved blocks of conservation that can be clearly observed in a sequence alignment. Often distantly related genes have domains conserved beyond the rest of the gene. Domains also will fold and evolve independently of the rest of the protein, resulting in increased conservation over the domain region. The idea of a protein domain was introduced by Wetlaufer in 1973 through observing common stable units in X-ray crystallography studies.

A protein domain can be understood as similar to a motif, although some databases have more sophisticated representations when necessary to describe a more complex pattern. To identify protein domains, one needs a sufficiently diverged set of proteins in order to accumulate enough mutations outside the domain so that the domain itself can be identified. However, with a set of proteins that is too diverged, once could even lose the conservation of the domain.

Protein Domain Annotation Databases

Pfam is a database of proteins, protein families, and domains, and is available at `http://pfam.sanger.ac.uk`. Pfam domains are a commonly used annotation of protein domains that are relatively easy to use.

For example, one can use the program HMMer (pronounced "hammer") to scan Pfam domains downloaded as an hmm file. The command would be something like this:

```
hmmscan --domtblout domainTable.txt Pfam-A.hmm proteins.fasta
```

To produce a table similar to a BLAST output.

Other databases for protein domains are:

- PANTHER
- InterPro
- PROSITE

8.4 Secondary Structure prediction

To get a full view of a protein's structure, we'll need a way to compute its tertiary structure, which is extremely difficult. In practice, we have to rely on experimental evidence such as X-ray crystallographic studies or NMR structures. Some studies will use a known structure of a protein and apply it to a homologous protein. Secondary structure for proteins can be computed, however, with reasonable accuracy. The secondary structure of a protein is a list of the positions corresponding to alpha helices and beta sheets.

For example, the program `jnet` can be run on the command line with a command like:

```
$ jnet -p human_catalase.fasta
```

to produce an output file indicating the locations of these regions:

```
Length = 527  Homologues =  1
 RES    : MADSRDPASDQMQHWKEQRAAQKADVLTTGAGNPVGDKLNVITVGPRGPLLVQDVVFTDEMAHFD
 ALIGN  : ---------HHHHHHHHHHHHHH--EEE---------EEEEEEE-----EEEEEEEE-----H--
 CONF   : 888888875257788899888741224406877757526788834777525754552010000 13
 FINAL  : ---------HHHHHHHHHHHHHH--EEE---------EEEEEEE-----EEEEEEEE--------
```

EXP	Inferred from Experiment
IDA	Inferred from Direct Assay
IPI	Inferred from Physical Interaction
IMP	Inferred Mutant Phenotype
IGI	Inferred from Genetic Interaction
IEP	Inferred from Expression Pattern
ISS	Inferred from Sequence or Structural Similarity
ISO	Inferred from Sequence Orthology

Table 8.1: Table of some GO term evidence codes and their meaning.

where the `CONF` value is the confidence, a number from 0 to 9, that indicates the quality of the prediction. In this representation, the `H`s represent alpha helices, and the `E`s indicate beta-strands. Here we have run the program with one protein sequence, but it could be run with multiple sequences for greater accuracy.

8.5 Gene Ontology

Gene Ontology (GO) is a database of controlled vocabulary terms that describe gene/protein function (`http://geneontology.org`). These terms form a hierarchy, where certain higher-level terms "contain" the lower-level terms in conceptual scope. There are many levels of evidence for the assignment of a GO term, indicated by "evidence codes", given by a symbol of a few letters. For example, `EXP` indicates that the term was "Inferred by experiment". Others such as `TAS` for "traceable author statment" indicate that there is a statement in a publication making the claim of the association.

8.5.1 Multiple hypothesis testing

Frequently when performing a study on the effect of a mutation or other treatment, one looks at the GO terms of the genes that are significantly differentially expressed due to the experimental input. One evaluates whether the enrichment of a particular GO term in the data exceeds that of the expected rate of this term occurring in randomly sampled gene sets of the same size. This significance is typically computed from a p-value describing the enrichment.

However, in doing so, one needs to perform a multiple test correction, to account for the fact that many hypotheses were tested. For example, if you found a one-in-a-million event after examining one million events, you might not be surprised. The simplest multiple-test correction is the Bonferroni correction, where when testing N hypotheses, we need to test for significance at level α by examples satisfying the relationship:

$$pN \leq \alpha \tag{8.1}$$

Another method is the Benjamini-Hochberg procedure, which is a little bit more complex. To do this procedure, one needs to sort the list of p-values $p_1, p_2, ..., p_N$ in ascending order such that $p_{(1)} \leq p_{(2)} \leq ... \leq p_{(N)}$ such that the subscript given by $p_{(r)}$ indicates the p-value with rank r. This procedure gives us a list of significant hypotheses with a rate of false-discovery defined by the imposed FDR, a number between 0 and 1. After the sorting, all p-values with a rank r less than the maximum rank $r*$ such that:

$$\frac{p_{(r*)}N}{r*} \leq FDR, \tag{8.2}$$

where FDR defines the false discovery rate, are deemed significant.

Chapter 9

Gene Regulation

Genes can be regulated in several different ways, and be regulated transcriptionally (at the gene-level), post-transcriptionally (at the mRNA level), translationally, and post-translationally. Splicing of mRNA can also be regulated and there are many other inputs to regulation still. Clearly, transcriptional regulation and post-transcriptional regulation are best suited for study by nucleic acid sequence bioinformatics because techniques such as ChIP-Seq and RNA-Seq can be used to measure the inputs and outputs of these modes of regulation. Here we will focus on regulation by transcription factors using ChIP-Seq and regulation by microRNAs using small RNA-Seq and RNA-Seq.

9.1 Transcription Factors and ChIP-Seq

Transcription factors (TFs) are proteins that bind to DNA and are capable of regulating the transcription of genes. TFs can be activators, which activate transcription of the genes they regulate, or repressors, which block the transcription of the genes they regulate.

9.1.1 ChIP-Seq Peak identification

ChIP-Seq measures the genome-wide binding of a TF by sequencing DNA fragments that are bound to the TF in a controlled experiment. Antibodies that recognize epitopes in the protein of interest are used to "pull-down" the protein. When cross-linking is used, such as by adding formaldehyde, everything sticks together, holding the protein to the DNA that it is bound to. After the antibodies and associated chromatin are precipitated out from the solution and isolated, they can be reverse-crosslinked, and then a DNA-extraction is performed. The fragments of DNA are then sequenced using deep sequencing. Despite best efforts to just isolate fragments directly bound the the TF of interest, a lot of spurious reads make it through the process as well. Therefore, we need to sequence a control sample without the antibody, either using the immunogloblulin (IgG) or just naked DNA (input), so that we can look for statistical enrichment of the reads that align to a particular genomic location from the ChIP-Sample compared to this control sample.

Short read alignment with `bowtie`

Once we have the the reads from the high-throughput sequencer, typically in a FASTQ file, we need to align to the genome. This can be achieved with `bowtie` and a bowtie index of the genome.

```
bowtie -m 1 -S -q hg38 TF_ChIPSeq.fastq TF_ChIPSeq_hg38_bowtie.sam
bowtie -m 1 -S -q hg38 DNA_input.fastq DNA_input_hg38_bowtie.sam
```

In this particular example, we are aligning the fastq files such that we are only retaining alignments that have one hit to the genome. This is done with the "`-m 1`" option. If one wanted to retain only reads that map to at most `M` hits, then we would use "`-m M`" where `M` is some integer. Recall that `hg38` is the "base" of the bowtie index of the genome. It is actually defined by a set of files, as described in section 6.2.

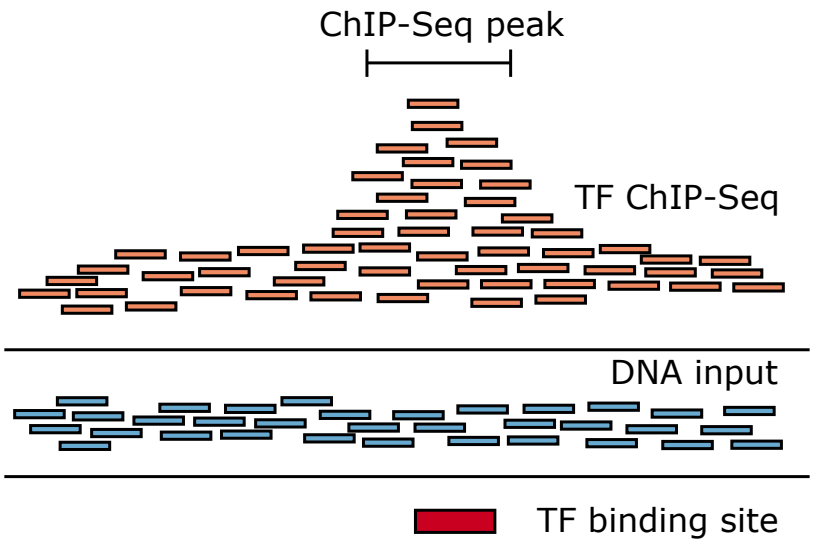

Figure 9.1: ChIP-Seq peaks are characterized by a local enrichment of ChIP-Seq reads compared to control, and are observed over the binding sites for the TF under investigation.

While on the subject of bowtie indexes, we can either download them from the web, such as from Illumina's iGenome datasets (`https://support.illumina.com/sequencing/sequencing_software/igenome.html`), or we can create one from a genome fasta file `hg38_genome.fasta` with the following command:

```
bowtie-build -f hg38_genome.fasta hg38
```

Identifying Peaks with PeakSeq

The software PeakSeq was one of the first pieces of software used to identify ChIP-Seq peaks using a statistical model. The program divides the genome into segments of length $L_{segment}$ such the number of reads aligning to each genomic segment i from the ChIP-Seq sample is $N_{ChIP}(i)$, along with the number of reads from the control sample $N_{control}(i)$. The statistical significance is assessed using a binomial model. That is, the reads for a particular segment i is assumed to be $N(i) = N_{ChIP}(i) + \alpha N_{control}(i)$ Bernoulli distributed random variables that take the value of $ChIP$ with a probability p and a value of *control* with a probability $q = 1 - p$. Then, it is determined whether the number of ChIP-Seq reads $N_{ChIP}(i)$ is larger than expected due to chance.

First, a best-fit line of slope α through the data for all segments is computed. The data that are fit is the x-axis has the value of $N_{control}$ and the y-axis has the value N_{ChIP}, and each data point is a segment. The best-fit line through the data gives a scale factor α that can be used to scale the number of reads $N_{control}$ for the control sample, to be compared to the number of reads for the ChIP sample. When the scale factor is used, the binomial model uses a probability of $p = \frac{1}{2}$ when comparing N_{ChIP} to $\alpha N_{control}$.

The p-value for a binomial variable is computed as one minus the cumulative distribution function for the binomial distribution $F(k, n, p)$, which gives the probability of observing less than or equal to k successes in n trials of a Bernoulli random variable with probability p. The p-value is computed by

$$\text{p-value} = 1 - F(N_{ChIP} - 1, N, p) = 1 - \sum_{i=0}^{N_{ChIP}-1} \binom{N}{j} p^j (1-p)^{N-j}$$

Each p-value is used in a Benjamini-Hochberg correction described in section 8.5.1.

Identifying Peaks with MACS

MACS is a software that can find ChIP-Seqs peaks similar to that of PeakSeq. The difference is that MACS models the length distribution of the ChIP-DNA fragments, which helps it locate binding sites more

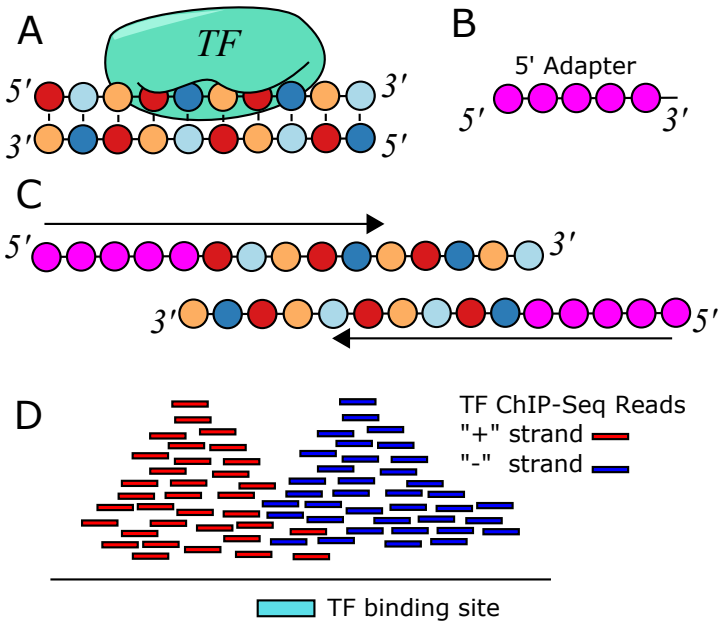

Figure 9.2: **A.** ChIP-Seq reads are sequenced from fragments bound to the TF. **B.** Reads are ligated to 5' adapters. C. Reads are sequenced from the 5' end due to how the adapters are attached. D. This results in a bimodal distribution of reads around the location of the TF binding site

accurately. There are different versions of MACS, but probably the most commonly used is `macs14`, which can be run to identify peaks with commands like

```
macs14 -t TF_ChIP_hg38.bam -c DNA_input_hg38.bam -f BAM -n TF_ChIP_vs_DNA_input -g hs -p 0.01
```

This program supports other input formats other than `bam`, but this is probably the most common. The `-g hs` command gives it the effective genome size. This can either be a number, or a two letter abbreviation for the more common model systems. Finally, the `-p 0.01` specifies a p-value threshold defining what peaks to include.

9.1.2 Chromatin Modifications

While transcription factors tend to have sharp, well-defined peaks, ChIP-Seq performed on antibodies that recognize chromatin modifications tend to be enriched for broad domains that can extend for many megabases. Certain chromatin marks, such as H3K27me3 can be spread out over very large polycomb-repressed domains. Some programs, such as SICER, are specially designed to identify these kinds of broad domains. Other programs such as chromHMM will train an HMM on a set of samples

9.2 MicroRNA regulation and Small RNA-Seq

MicroRNAs are processed from endogenous primary transcripts called "pri-miRs" that can either be transcribed as a noncoding RNA or as the the intron of a protein-coding gene. The base of one or more specific hairpin sequences in the pri-miR are processed out by Drosha, resulting in a microRNA hairpin precursor called a "pre-miR". The pre-miR is exported out of the nucleus, and then the loop of the hairpin is removed by Dicer, which is an RNase III enzyme. The result is an RNA duplex of two mature microRNA sequences. The resulting mature microRNAs (miRs) are small (18-25nt) RNA molecules that regulate gene expression

throughout the eukaryotes. The word for microRNAs is typically written in lower-case as in "microRNAs" except when at the beginning of the sentence. The regulation by miRs is post-transcriptional, and can either lead to degradation of the transcript it regulates, or translational inhibition. MicroRNAs operate through complementary sequences in the 3' UTRs of transcripts that match the "seed" sequence, defined as positions 2-8 of the mature microRNA. That is, the seed sequence of the miR is complementary to the target site in the 3' UTR of the gene it regulates.

9.2.1 Read abundance

The first thing we might be interested in knowing about a microRNA is the extent to which its mature miR is expressed. This can be done with small RNA-Seq, which is RNA-Seq performed on size-selected fragments. When aligning microRNA reads, we first have to trim the 3' adapters from the FASTQ files as explained in section 6.8.

```
bowtie -m 50 -l 20 -n 2 -S -q hg38 smallRNASeq.fastq smallRNA_hg38_bowtie.sam
```

To compare across samples, it is best to use a normalized expression count, such as Reads Per Million (RPM), which allows expression levels to be compared between samples of varying depth. For microRNA m with the number of reads mapped to it being R_m, the RPM is computed by

$$RPM_m = \frac{R_m 10^6}{N}$$

where N is the number of reads in the sample. This equation is similar to RPKM from equation 6.1, but doesn't normalize per length. This is because microRNAs always have reads spanning the full locus defining the mature microRNA when mapped to the genome, so being a few nucleotides longer doesn't give one mature miR annotation an advantage over a shorter one. Also, normalizing by length can artificially inflate differences between a short miR of length $18nt$ and a longer miR of length $25nt$ (both lengths exist in microRNA databases) even when they both have the same number of reads.

9.2.2 Argonaut CLIP-Seq

To directly measure functional mature microRNAs that are actually incorporated into the RISC complex, one can perform AGO-IP, or an immunoprecipitation with an antibody for Argonaut (AGO).

9.2.3 MicroRNA target prediction

There are many databases that contain information on microRNA target sites, such as `http://www.microrna.org/microrna/getGeneForm.do`, which allows you to retrieve alignments of 3' UTRs and associated target sites.

The software TargetScan allows one to predict target sites from a set of miR sequences and a set of UTR sequences. The format is pretty simple, but because of the information used, a simple FASTA file does not suffice. An additional column is required for the taxa ID of the species used for each example. For the microRNAs, the file format contains a tab-delimited list of miR-Family names, seed sequences, and taxa ID:

```
let-7/98        GAGGUAG        10090
let-7/98        GAGGUAG        10116
let-7/98        GAGGUAG        9031
let-7/98        GAGGUAG        9606
let-7/98        GAGGUAG        9615
miR-18        AAGGUGC        10090
miR-18        AAGGUGC        10116
miR-18        AAGGUGC        9031
miR-18        AAGGUGC        9606
miR-18        AAGGUGC        9615
miR-1/206        GGAAUGU        10090
miR-1/206        GGAAUGU        10116
miR-1/206        GGAAUGU        9031
miR-1/206        GGAAUGU        9606
miR-1/206        GGAAUGU        9615
```

For example, in this example 9606 is the taxa ID for human. These taxa IDs are designated by NCBI Taxonomy, at `http://www.ncbi.nlm.nih.gov/taxonomy`. The 3' UTRs are kept in a similar file that contains sequences from a multiple sequence alignment of the UTR sequences. The UTR file also has taxa IDs for each sequence:

```
CDC2L6  10090   CCCACUCCCU---CU-------------------------GCUUGGCCUUGGA-----------CUCCAGCAGGGUGGUAUUUGUGUUAC---AAAGACCCCCAG...
CDC2L6  10116   CCCACUGCCC-----------------------------GCUUGGCCUCGGGGAGCACAGAGCCGGCGGCAGGGUGGUUUCUGCAAUAC---AAAGAACCCCAC...
CDC2L6  9031    GAGGCUUAACAGACUGCACUGAAAAAGGAAUGGAUUAAAAGCCAGA----AGA-----------CUCCAGCAAUAUGAAGUUCGUGUUGAUGAGAAGAACCAAA-...
CDC2L6  9606    CCAGCUCCCG---UUGGGCCAGGCCAG--------CCCAGCCCAGAGCACAGG-----------CUCCAGCAAUAUG---UCUGCAUUGA---AAAGAACCAAAA...
CDC2L6  9615    CCG-CCACGG-------------------------------CCCAGGGCACAGA-----------CUCCAGCAAUAUGAUGUCCGCAUUGA---CAAGAACCAAA-...
FNDC3A  10090   AAAUAUAACUUUAUUUUUAA-CACU--GUAUUACAUUUUAUUUUGUCAUGUACU-AAAAUUAUUUCUGUA---UUGCUUUUAC--AAAAUGGUGGCAUUUAGCAC...
FNDC3A  10116   AAAUAUAACUUUAUUUUUAA-CAC----UAUUACAUUUUAUUUUGUCAUGUACU-AAAAUUAUUUCUGUA---UUGCUUUUAC--AAAAUGGUGGCAUUUAGCAC...
FNDC3A  9031    AGAAACAGAUUUAUUUUAGAAUGCUGCCCAUUACAUUUUACUUUCUCAUAAUCUAAAAAAAAAUUCUGUUCUCUUGCUUUUACAAA-AACA--GGCAUUUAGCAC...
FNDC3A  9606    AAAUAUAACUUUAUUUUUUA--ACU--CUAUUACAUUUUAUUUUGUCAUGUACU-AAAAUUAUUUCUGUA---UUGCUUUUAUAAAAAACAGUGGCAUUUAGCAC...
FNDC3A  9615    AAAUAUAACUUUAUUUUUAA-UACU--GUAUUACAUUUUAUUUUGUCAUGUACU-AAAAUUAUUUCUGUA---UUGCUUUUAC-AAAAACAGUGGCAUUUAGCAC...
```

These two examples are in fact the first few lines if the sample files given with the TargetScan software. TargetScan is a database of conserved microRNA target sites, but also is a set of software that can be used to predict conserved target sites from two files like these.

```
perl targetscan_70.pl miR_Family_info_sample.txt UTR_Sequences_sample.txt targetscan_output.txt
```

Appendix A

Mathematical Preliminaries

Some students with limited mathematical background may benefit from a brief introduction to some common mathematical notation frequently used in this text.

An efficient way to write a sum of many variables is with the sigma notation $\sum$. Consider adding up the values $x_1 + x_2 + x_3 + x_4 + x_5$. This can be compactly represented as:

$$\sum_{k=1}^{5} x_k = x_1 + x_2 + x_3 + x_4 + x_5$$

Similarly, if we want to represent a product of many variables, we can use the following notation:

$$\prod_{k=1}^{n} x_k = x_1 \times x_2 \times ... \times x_n$$

An example of such a product function is "factorial", $n!$. This quantity is the produce of the integers from 1 to n, and represents the number of possible arrangements of n distinct objects.

$$n! = \prod_{k=1}^{n} k$$

Logarithms are an important function to know in bioinformatics as they are commonly used in scoring systems. Logarithms are powerful because they have the following useful algebraic property (among others):

$$\log(AB) = \log(A) + \log(B)$$

We can combine the properties of our sums and products and logs, with the following equation:

$$\log\left(\prod_{k=1}^{n} x_k\right) = \sum_{k=1}^{n} \log\left(x_k\right).$$

Another very useful mathematical construct is the Kronecker Delta function, and can be used for counting. It is defined as follows:

$$\delta_{a,b} = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{if } a \neq b \end{cases} \tag{A.1}$$

Appendix B

Probability

B.0.4 Probability Distributions

Probability Mass Function

A probability mass function for a discrete random variable k is defined such that $P(k)$ is the probability of the variable taking the value k. These distributions are normalized, meaning summing over all possible values will equal 1. In other words:

$$\sum_k P(k) = 1$$

Probability Density Function

For continuous random variables x, a Probability Density Function gives the value $f(x)$ corresponding to the likelihood of the random variable taking on that value, and probabilities are computed from

$$P(a \leq x \leq b) = \int_a^b f(x)dx$$

Integrating over all values of x will equal 1.

$$\int_{-\infty}^{\infty} f(x)dx = 1$$

B.0.5 Conditional Probability

In probability, we often want to talk about conditional probability, which gives us the probability of an event given that we know that another event has occurred.

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Bayes' Theorem tells us that:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

We can also expand a probability of an event into the conditional probabilities of each other condition, times the probability of these other events.

$$P(x) = \sum_{i=1}^{N} P(x|A_i)P(A_i)$$

This expression $P(x)$ is often called the evidence. We can express the probability of A given x, which is called the posterior probability by

$$P(A|x) = \frac{P(x|A)P(A)}{P(x)}$$

Also in this equation, $P(x|A)$ is called the likelihood, and $P(A)$ is called the prior.

B.0.6 Bernoulli Distribution

One of the simplest probability distributions we should consider is the Bernoulli Distribution. It is essentially a single event, with a probability of success denoted p, and a probability of failure of $q = 1 - p$.

Example 1: A coin toss. Heads or tails? Typically for a fair coin, this will be such that $p = q = 0.5$.

Example 2: Selecting a single nucleotide from the genome at random. Is it purine (A or G) or is it pyrimidine (C or T)? For the human genome, the GC content is about 0.417, but it varies from chromosome to chromosome.

B.0.7 Binomial Distribution

Central to the Binomial distribution is the binomial coefficient $\binom{n}{k} = \frac{n!}{(n-k)!k!}$ The number of ways to have k successes out of n trials is $\binom{n}{k}$. The probability of observing k successes out of n trials is $P(k|p,n) = \binom{n}{k}p^k q^{n-k}$. Since $p + q = 1$, and 1 raised to the nth power is still 1, we can see that these probabilities are normalized (sum to one) by the equation:

$$1 = (p+q)^n = \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k}$$

Example 1: Consider tossing a fair coin 100 times, and repeating this for 1000 trials. A histogram for k, number of heads in n tosses is shown in Figure B.1. The expected value of k is $E[k] = np$, and the variance is $var(k) = npq$.

B.0.8 Multinomial Distribution

The multinomial distribution deals which situations when there are more than two possible outcomes (for example, DNA nucleotides). The multinomial coefficient, $M(\vec{n})$, in this example describes the number of ways to have a sequence of length n, with the number of occurrences of A,C,G,and T to be n_A, n_C, n_G, and n_T respectively.

$$M(\vec{n}) = \frac{n!}{n_A! n_C! n_G! n_T!}$$

The probability of a particular set of observed counts $\vec{n} = (n_A, n_C, n_G, n_T)$ depends on the frequencies $\vec{p} = (p_A, p_C, p_G, p_T)$ by the expression:

$$P(\vec{n}|\vec{p}) = \frac{n!}{n_A! n_C! n_G! n_T!} \prod_{i=A}^{T} p_i^{n_i}$$

B.0.9 Poisson Distribution

The Poisson Distribution describes the number of observed events given an expected number of occurrences λ. Consider, for example, the probability of a red car driving past a particular street corner in your neighborhood. Its probability mass function is given by:

$$P(k|\lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$$

It has one parameter λ, which is also the expected value of k and the variance ($E[k] = var(k) = \lambda$).

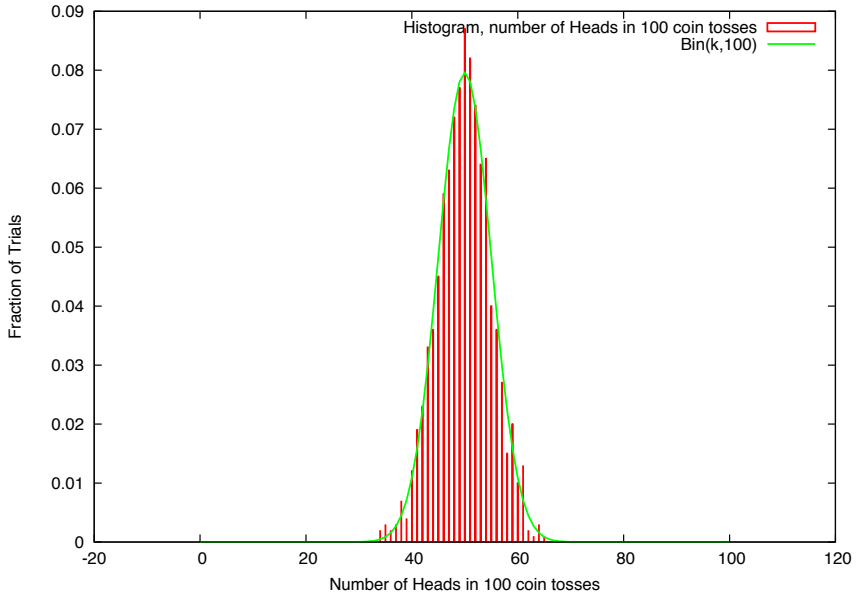

Figure B.1: A histogram of the number of heads observed in 100 tosses of a fair coin, repeated 1000 times.

It can be shown that for fixed $\lambda = np$, and for $n \to \infty$, the Binomial distribution is equivalent to the Poisson distribution.

In many applications, the λ parameter for the Poisson distribution is treated as a rate. In this case, the expected value of k is $E[k] = \lambda\ell$, so that λ describes the rate of occurrence of the event per unit time (or per unit distance, or whatever the application is). The probability distribution is given by:

$$P(k, \ell|\lambda) = \frac{(\lambda\ell)^k e^{-\lambda\ell}}{k!}$$

Example: Consider modeling the number of mutations k observed in a length of DNA ℓ.
Consider the important case when $k = 0$ for the Poisson Distribution.

$$P(k = 0, \ell|\lambda) = e^{-\lambda\ell}$$

The probability that k is not zero, is therefore:

$$P(k > 0, \ell|\lambda) = 1 - e^{-\lambda\ell} \tag{B.1}$$

Now consider ℓ being replaced by a random variable x.

B.0.10 Exponential Distribution

The Exponential distribution is related to the Poisson by equation $B.1$. It is the probability of an event occurring in a length x. The cumulative distribution function $F(x)$ is the same as this last equation:

$$F(x|\lambda) = 1 - e^{-\lambda x}$$

And the probability density function is the derivative of the cumulative distribution function:

$$P(x|\lambda) = \lambda e^{-\lambda x}$$

Note that whereas the Poisson distribution is a discrete distribution, the Exponential distribution is continuous and x can take on any real value such that $x \geq 0$.

B.0.11 Normal Distribution

A very important distribution for dealing with data, the normal distribution models many natural processes. We already saw that the binomial distribution looks like the normal distribution for large n. Indeed, the Central Limit Theorem, the sum of random variables approaches the normal distribution for large n. Here is the p.d.f.:

$$P(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

B.0.12 Extreme Value Distribution

The maximum value in a set of random variables $X_1, ..., X_N$ is also a random variable. This variable is described by the Extreme Value Distribution, also known as the Gumbel Distribution.

$$F(x|\mu,\sigma,\xi) = \exp\left\{-\left[1+\xi\left(\frac{x-\mu}{\sigma}\right)\right]^{-1/\xi}\right\}$$

For $1+\xi(x-\mu)/\sigma > 0$, and with a location parameter μ, a location parameter σ, and a shape parameter ξ. In the limit that $\xi \to 0$, the distribution reduces to:

$$F(x|\mu,\sigma) = \exp\left\{-\exp\left(-\frac{x-\mu}{\sigma}\right)\right\}$$

In many cases, for practical purposes, one makes the assumption that $\xi \to 0$.

Bibliography

[1] F.H.C. Crick. On protein synthesis. *Symp. Soc. Exp. Biol*, XII:139–163, 1956. `http://profiles.nlm.nih.gov/SC/B/B/F/T/_/scbbft.pdf`.

[2] F. Crick. Central dogma of molecular biology. *Nature*, 227(5258):561–563, Aug 1970.

[3] David J Lipman and William R Pearson. Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441, 1985.

[4] Brent Ewing and Phil Green. Base-calling of automated sequencer traces using phred. ii. error probabilities. *Genome research*, 8(3):186–194, 1998.

[5] Michael J Guertin and John T Lis. Chromatin landscape dictates hsf binding to target dna elements. *PLoS Genet*, 6(9):e1001114, 2010.

[6] T Tatusova, M DiCuccio, A Badretdin, V Chetvernin, S Ciufo, and W Li. The ncbi handbook. 2013.

[7] James Ostell and Eric W Sayers. Dennis a. benson, ilene karsch-mizrachi, karen clark, david j. lipman [j]. *Nucleic acids research*, 1:6, 2011.

[8] Kim D Pruitt, Tatiana Tatusova, and Donna R Maglott. Ncbi reference sequences (refseq): a curated non-redundant sequence database of genomes, transcripts and proteins. *Nucleic acids research*, 35(suppl 1):D61–D65, 2007.

[9] Donald E. Knuth, Jr. James H. Morris, and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977.

[10] Michael Burrows and David J. Wheeler. A block sorting lossless data compression algorithm. *Technical Report 124, Digital Equipment Corporation*, May 1994.

[11] Andrew D Johnson. An extended iupac nomenclature code for polymorphic nucleic acids. *Bioinformatics*, 26(10):1386–1389, 2010.

[12] Thomas W Burke and James T Kadonaga. Drosophila tfiid binds to a conserved downstream basal promoter element that is present in many tata-box-deficient promoters. *Genes & development*, 10(6):711–724, 1996.

[13] John C Wootton and Scott Federhen. [33] analysis of compositionally biased regions in sequence databases. *Methods in enzymology*, 266:554–571, 1996.

[14] G. D. Stormo, T. D. Schneider, L. Gold, and A. Ehrenfeucht. Use of the 'Perceptron' algorithm to distinguish translational initiation sites in E. coli. *Nucleic Acids Res.*, 10(9):2997–3011, May 1982.

[15] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[16] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[17] Anthony Mathelier, Xiaobei Zhao, Allen W Zhang, François Parcy, Rebecca Worsley-Hunt, David J Arenillas, Sorana Buchman, Chih-yu Chen, Alice Chou, Hans Ienasescu, et al. Jaspar 2014: an extensively expanded and updated open-access database of transcription factor binding profiles. *Nucleic acids research*, page gkt997, 2013.

[18] Gavin E Crooks, Gary Hon, John-Marc Chandonia, and Steven E Brenner. Weblogo: a sequence logo generator. *Genome research*, 14(6):1188–1190, 2004.

[19] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6):721–741, 1984.

[20] Timothy L Bailey, Charles Elkan, et al. Fitting a mixture model by expectation maximization to discover motifs in bipolymers. 1994.

[21] Dayhoff M.O., R.M. Schwartz, and B.C. Orcutt. A model of evolutionary change in proteins. *Atlas of Protein Sequence and Structure*, 5(3):345–352, 1978.

[22] Steven Henikoff and Jorja G Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, 1992.

[23] Samuel Karlin and Stephen F Altschul. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proceedings of the National Academy of Sciences*, 87(6):2264–2268, 1990.

[24] Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

[25] I. Jonkers and J. T. Lis. Getting up to speed with transcription elongation by RNA polymerase II. *Nat. Rev. Mol. Cell Biol.*, 16(3):167–177, Mar 2015.

[26] M. F. Lin, I. Jungreis, and M. Kellis. PhyloCSF: a comparative genomics method to distinguish protein coding and non-coding regions. *Bioinformatics*, 27(13):i275–282, Jul 2011.

[27] L. Wang, H. J. Park, S. Dasari, S. Wang, J. P. Kocher, and W. Li. CPAT: Coding-Potential Assessment Tool using an alignment-free logistic regression model. *Nucleic Acids Res.*, 41(6):e74, Apr 2013.

[28] M. Guttman, P. Russell, N. T. Ingolia, J. S. Weissman, and E. S. Lander. Ribosome profiling provides evidence that large noncoding RNAs do not encode proteins. *Cell*, 154(1):240–251, Jul 2013.

[29] A. A. Bazzini, T. G. Johnstone, R. Christiano, S. D. Mackowiak, B. Obermayer, E. S. Fleming, C. E. Vejnar, M. T. Lee, N. Rajewsky, T. C. Walther, and A. J. Giraldez. Identification of small ORFs in vertebrates using ribosome footprinting and evolutionary conservation. *EMBO J.*, 33(9):981–993, May 2014.