Chapter 9

Sequence Analysis

A common task in bioinformatics is to find sequence motifs within or similarites between different nucleotide or amino acid sequences. The reason being that similar sequences tend to have similar biological functions. To quantify the similarity of sequences, it is necessary to compute first their optimal alignment. In this chapter, we illustrate how global and local optimal pairwise alignments can be obtained. Other important sequence-specific quantities for DNA sequences include the GC fraction, and for amino acid sequences, the isoelectric point and hydropathy score. We will illustrate how to compute these quantities as well. In addition, in this chapter we will cover how to query online databases, translate RNA into protein sequences, and perform pattern matching within a sequence.

9.1 Using a query language

We will be using the query tools available in the `seqinr` package to perform various types of searches. However, before we download anything it is important to know which databanks can be chosen for querying by calling the `choosebank()` function:

```
> library(seqinr)
> choosebank()
 [1] "genbank"         "embl"            "emblwgs"         "swissprot"
 [5] "ensembl"         "hogenom"         "hogenomdna"      "hovergendna"
 [9] "hovergen"        "hogenom5"        "hogenom5dna"     "hogenom4"
[13] "hogenom4dna"     "homolens"        "homolensdna"     "hobacnucl"
[17] "hobacprot"       "phever2"         "phever2dna"      "refseq"
[21] "greviews"        "bacterial"       "protozoan"       "ensprotists"
[25] "ensfungi"        "ensmetazoa"      "ensplants"       "ensemblbacteria"
```

```
[29] "mito"             "polymorphix"      "emglib"           "taxobacgen"
[33] "refseqViruses"
```

Note that if you do not see results similar to above, then it is highly likely that you are behind a firewall that is blocking the TCP/IP port used to connect to the `seqinr` server - which is port 5558.

The results from the `choosebank()` function show that many different databases containing lots of different types of data are available through the `query()` interface. In this chapter, we will give a few examples to illustrate some of the most important query capabilities. Note that sometimes we will use the `virtual=TRUE` option to save time by preventing any downloading of the actual sequences.

Example1: Number of available sequences. First, we can query for the number of CCND3 sequences in Genbank across all species:

```
> choosebank("genbank")
> query("ccnd3","k=ccnd3",virtual=TRUE)$nelem
[1] 24
```

More specifically, we can query for the number of CCND3 sequences in Genbank for the species homo sapiens:

```
> query("ccnd3.human","sp=homo sapiens AND k=ccnd3",virtual=TRUE)$nelem
[1] 6
```

Table 9.1 lists the selection criteria available for performing a query through the `seqinr` package. Note that the criteria key can be in lower or upper case. However, like the examples above, there can be no spaces between the criteria key and the value.

For many other combinations of search options using AND, OR, and NOT, we refer to the manual of the `seqinr` package, and for a book length treatment with many examples to Charif et al. (2008).

9.2 Getting sequence information

After sequences are downloaded in binary format, it's also important to obtain information with respect to their accession number, length, number of elements, and other annotations. We illustrate how to do this with an example.

Example 1: CCND3-like genes. Let's download sequences related to the species homo sapiens and a gene name like "CCND3".

Table 9.1: Query selection criteria (no space before the = sign).

Criteria	Description
SP=taxon	Seqs attached to taxon or any other below in tree; @ = wildcard
TID=id	Seqs attached to given numerical NCBI's taxon id
K=keyword	Seqs attached to keyword or any other below in tree; @ = wildcard
T=type	Seqs of specified type
J=journalname	Seqs published in journal specified using defined journal code
R=refcode	Seqs from reference specified jcode/volume/page (e.g., JMB/13/5432)
AU=name	Seqs from references having specified author (only last name)
AC=accessionno	Seqs attached to specified accession number
N=seqname	Seqs of given name (ID or LOCUS); @ wildcard possible
Y=year	Seqs published in specified year; > and < can be used instead of =
O=organelle	Seqs from specified organelle (e.g., chloroplast)
M=molecule	Seqs from specified molecule in ID or LOCUS annotation
ST=status	Seqs from specified data class (EMBL) or review level (UniProt)

```
> choosebank("genbank", timeout=30)
> ccnd3.human = query("ccnd3.human","sp=homo sapiens AND k=ccnd3@")
> ccnd3.human$nelem
[1] 7
```

The 9 sequences are downloaded in binary format. The symbol `@` acts as a wildcard for zero or more characters. There are a number of useful functions available to obtain further information. These include `getName()`, `getKeywords()`, `getLength()`, `getSequence()`, `getTrans()`, and `getAnnot()`. To use these functions on a list containing sets of sequences, the `sapply()` function is very convenient. We illustrate this by extracting the keywords and NCBI accession numbers:

```
> sapply(ccnd3.human$req, getKeyword)
[[1]]
[1] "CYCLIN D3" "CCND3"     "AAM51826"

[[2]]
[1] "CYCLIN D3" "CCND3"     "AAH11616"

[[3]]
[1] "DIVISION PRI"
[2] "FULL ORF SHUTTLE CLONE, GATEWAY(TM), COM"
[3] "SOURCE"
[4] "GENE"
[5] "CAG47042"
[6] "CCND3"
[7] "CDS"
[8] "RELEASE 167"
```

```
[[4]]
[1] "D3-TYPE CYCLIN" "CCND3"          "AAA51927"

[[5]]
[1] "PSEUDO" "CCND3P"

[[6]]
[1] "PARTIAL"   "CYCLIN D3" "CCND3"     "AAA51929"

[[7]]
[1] "CYCLIN D3" "CCND3"     "AAA52137"

> sapply(ccnd3.human$req, getName)
[1] "AF517525.CCND3"   "BC011616.CCND3"   "CR542246"         "HUMCCND3A.CCND3"
[5] "HUMCCND3PS.PE1"   "HUMCCNDB04.CCND3" "HUMCYCD3A.CCND3"
```

Similarly, the length of the sequences can be obtained with the `getLength()` function:

```
> sapply(ccnd3.human$req, getLength)
[1] 879 879 879 879 537 559 879
```

Let's obtain the first sequence and print its first 15 nucleotides to the screen. Remember, we use double brackets to extract a sequence from a list:

```
> getSequence(ccnd3.human$req[[1]])[1:15]
 [1] "a" "t" "g" "g" "a" "g" "c" "t" "g" "c" "t" "g" "t" "g" "t"
```

Also, the translation into amino acids can be obtained:

```
> getTrans(ccnd3.human$req[[1]])[1:15]
 [1] "M" "E" "L" "L" "C" "C" "E" "G" "T" "R" "H" "A" "P" "R" "A"
```

Next, we extract the annotation information for the first sequence in the list:

```
> getAnnot(ccnd3.human$req[[1]])
 [1] "  CDS join(1051..1248,2115..2330,5306..5465,6005..6141,"
 [2] "  6593..6760)"
 [3] "  /gene=\"CCND3\""
 [4] "  /codon_start=1"
 [5] "  /product=\"cyclin D3\""
 [6] "  /protein_id=\"AAM51826.1\""
 [7] "  /db_xref=\"GI:21397158\""
 [8] "  /translation=\"MELLCCEGTRHAPRAGPDPRLLGDQRVLQSLLRLEERYVPRASY"
 [9] "  FQCVQREIKPHMRKMLAYWMLEVCEEQRCEEEVFPLAMNYLDRYLSCVPTRKAQLQLL"
[10] "  GAVCMLLASKLRETTPLTIEKLCIYTDHAVSPRQLRDWEVLVLGKLKWDLAAVIAHDF"
[11] "  LAFILHRLSLPRDRQALVKKHAQTFLALCATDYTFAMYPPSMIATGSIGAAVQGLGAC"
[12] "  SMSGDELTELLAGITGTEVDCLRACQEQIEAALRESLREAAQTSSSPAPKAPRGSSSQ"
[13] "  GPSQTSTPTDVTAIHL\""
```

9.3 Computations on sequences

Two basic sequence-specific quantities to compute are the mononucleotide and the dinucleotide frequencies.

Example 1: Frequencies of nucleotides and dinucleotides. We shall continue with the first result from the CCND3 (Cyclin D3) search with accession number "AF517525.CCND3". To compute the mononucleotide frequencies, we extract the first sequence from the list and then call the `table()` function:

```
> table(getSequence(ccnd3.human$req[[1]]))
  a   c   g   t
162 288 267 162
```

This table can also be computed with the `seqinr` function `count()`, which is more general in the sense that frequencies of dinucleotides can also be computed:

```
> count(getSequence(ccnd3.human$req[[1]]),2)
 aa  ac  ag  at  ca  cc  cg  ct  ga  gc  gg  gt  ta  tc  tg  tt
 25  44  64  29  68  97  45  78  52 104  76  34  16  43  82  21
```

In fact, the counts for any size *k*-mers can be calculated. For example, changing the 2 to 3 in the function call to `count()` above makes it possible to count trinucleotides.

Example 2: GC content. We can also compute the fraction of Gs or Cs (`GC content`) in a DNA sequence using the `GC()` function. There are similar functions that calculate `GC content` for just the first codon position in an open reading frame (`GC1()`), the second position (`GC2()`), and the third position (`GC3()`):

```
> GC(getSequence(ccnd3.human$req[[1]]))
[1] 0.6313993
> GC1(getSequence(ccnd3.human$req[[1]]))
[1] 0.6484642
> GC2(getSequence(ccnd3.human$req[[1]]))
[1] 0.4641638
> GC3(getSequence(ccnd3.human$req[[1]]))
[1] 0.78157
```

Hence, for the first sequence the GC content is largest in the third codon position. It is also possible to compute the G + C fraction in a sliding window along the sequence. Below we calculate the GC content for sliding windows of length 50 and plot them along the sequence:

```
> require(zoo)
> seq = getSequence(ccnd3.human$req[[1]])
> gcContent = rollapply(seq, width=50, by=1, FUN=GC, align="left") # find the 50bp-windowed
    ↪  running GC content fraction
> plot(gcContent,type="l")
```

From the resultant Figure 9.1, we can see that the GC content within the 50-nucleotide sliding windows changes drastically along the 800+ nucleotide gene sequence.

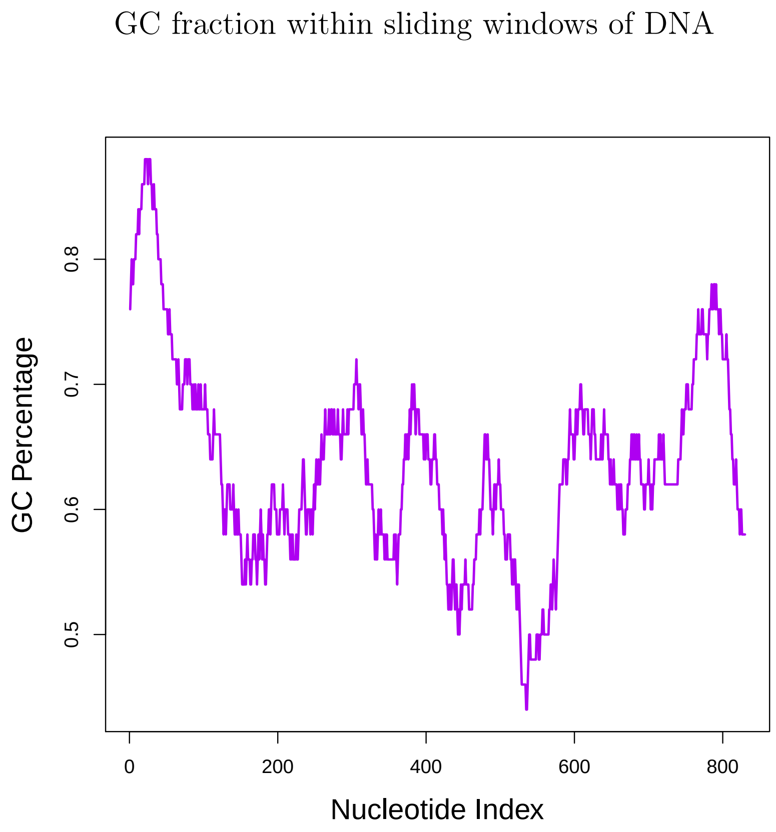

Figure 9.1: GC fraction within 50-nucleotide sliding windows along the DNA sequence for gene "AF517525.CCND3".

Example 3: Rho and z-scores. In relation to over or under-representation of dinucleotides, there is a function ρ (rho) which is defined as:

$$\rho(xy) = \frac{f_{xy}}{f_x \cdot f_y},$$

where f_{xy}, f_x, and f_y are the frequencies of the dinucleotide xy, mononucleotide x, and mononucleotide y, respectively. The rho scores indicate which dinucleotides are over or under-represented after normalizing for the mononucleotide content.

The dinucleotide z-scores are computed by subtracting the mean dinucleotide frequency and dividing by the standard deviation across all the dinucleotide frequencies. (Palmeira, et al., 2006).

The coefficients rho and the corresponding z-scores will be computed from the sequence with NCBI accession number "AF517525.CCND3":

```
> round(rho(getSequence(ccnd3.human$req[[1]])),2)

  aa   ac   ag   at   ca   cc   cg   ct   ga   gc   gg   gt   ta   tc   tg   tt 
0.84 0.83 1.30 0.97 1.28 1.03 0.51 1.47 1.06 1.19 0.94 0.69 0.54 0.81 1.67 0.70 

> round(zscore(getSequence(ccnd3.human$req[[1]]),modele='base'),2)

   aa    ac    ag    at    ca    cc    cg    ct    ga    gc    gg    gt    ta 
-1.08 -1.67  2.81 -0.18  2.78  0.42 -6.63  4.64  0.54  2.60 -0.80 -2.87 -3.10 
   tc    tg    tt 
-1.86  6.22 -1.98 
```

Both the rho and the z-scores indicate that the CpG dinucleotide, with a rho score of 0.51 and a z-score of -6.63, is highly under-represented in the gene sequence given that it's a very GC-rich gene.

Example 4: Comparing amino acid frequencies. After we have translated the nucleotide sequence into its corresponding amino acid sequence, it's often useful to construct a plot indicating their amino acid frequencies. For example, amino acid frequencies can often give a good first impression about sequence similarity.

We continue with the first result from the CCND3 (Cyclin D3) search, translate it into protein sequence, and then produce a `dotchart()` showing the amino acid frequencies:

```
> table <- table(getTrans(ccnd3.human$req[[1]]))
> orderedTable <- table[order(table)]
> names(orderedTable) <- aaa(names(orderedTable))
> dotchart(orderedTable,pch=19,xlab="Stop and amino-acid-counts")
> abline(v=1,lty=2)
```

The above code was run on both the AF517525.CCND3 and AL160163.CCND3 sequences resulting in Figure 9.2 and 9.3, respectively. The two dotcharts show that the two sequences are highly similar with respect to amino acid content.

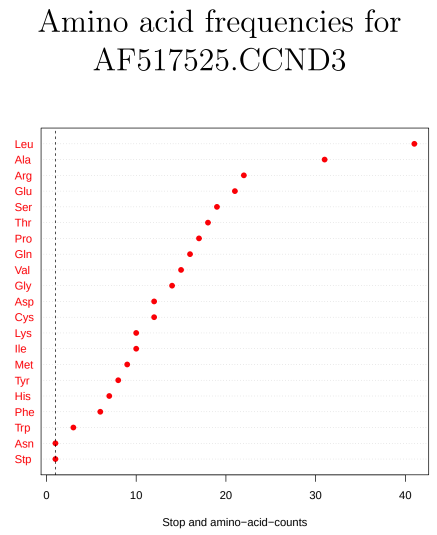

Figure 9.2: Frequency plot of amino acids for accession number AF517525.CCND3.

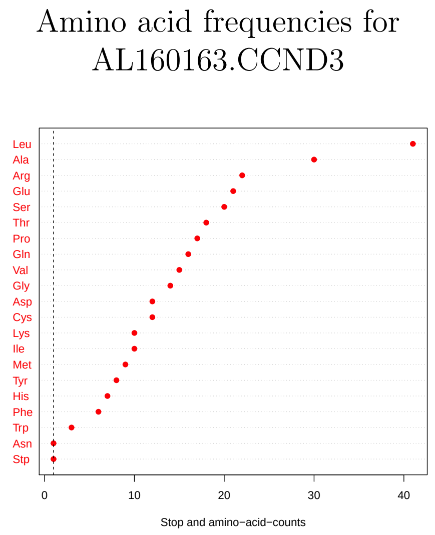

Figure 9.3: Frequency plot of amino acids for accession number AL160163.CCND3.

Example 5: Isoelectric point and molecular weight. When comparing amino acid sequences, it can also be useful to compute the theoretical isoelectric point or the molecular weight of the corresponding proteins. The function `computePI()` computes the theoretical isoelectric point of a protein, which is the pH at which the protein has a neutral charge (Gasteiger, et al. 2005):

```
> computePI(getTrans(ccnd3.human$req[[1]]))
[1] 6.657579
```

The protein molecular weight can be computed as follows:

```
> pmw(getTrans(getSequence(ccnd3.human$req[[1]])))
[1] 32503.38
```

Example 6: Hydropathy score. An important characteristic of a protein is its hydropathy score (Kyte & Doolittle, 1982), which is defined as a weighted sum $\sum_{i=1}^{20} \alpha_i f_i$ of amino acid coefficients α_i times the relative frequencies f_i. We'll present an example of how it can be computed.

The coefficients $\alpha_1, \cdots, \alpha_{20}$ are available as the `KD` data from the `EXP` list of the `seqinr` package. Below, the unique names are lexicographically ordered and stored in the object `kdc`. The sign of the scale is changed so that the scores of hydrophilic proteins are positive but smaller than one. Lastly, we define a function to compute the hydropathy score for a set of amino acid sequences:

```
> ccnd3.human.DNAseqs <- sapply(ccnd3.human$req, getSequence)
> ccnd3.human.AAseqs <- sapply(ccnd3.human.DNAseqs, getTrans)
> data(EXP)
> names(EXP$KD) <- sapply(words(), function(x) translate(s2c(x)))
> kdc <- EXP$KD[unique(names(EXP$KD))]
> kdc <- -kdc[order(names(kdc))]
>
> hydropathy <- function(data, coef) {   #data are sequences
+  f <- function(x) {
+      freq <- table(factor(x, levels = names(coef)))/length(x)
+      return(coef %*% freq) }
+  res <- sapply(data, f)
+  names(res) <- NULL
+  return(res)
+ }
> kdAth <- hydropathy(ccnd3.human.AAseqs, kdc)
> print(kdAth,digits=3)
 [1] 0.0874 0.0962 0.0189 0.1496 0.0962 0.0874 0.0874 0.2659 0.2220
```

Therefore, the largest score is still much smaller than one, so the conclusion is that there are no hydrophilic proteins among our sequences.

Also, the data set `aaindex` in the `seqinr` library contains more than five hundred sets of coefficients for computing sequence-specific quantities on protein sequences:

```
> data(aaindex)
> aaindex$CASG920101$D # description
[1] "Hydrophobicity scale from native protein structures (Casari-Sippl, 1992)"
> aaindex$CASG920101$I # inedxed data
 Ala  Arg  Asn  Asp  Cys  Gln  Glu  Gly  His  Ile  Leu  Lys  Met  Phe  Pro  Ser
 0.2 -0.7 -0.5 -1.4  1.9 -1.1 -1.3 -0.1  0.4  1.4  0.5 -1.6  0.5  1.0 -1.0 -0.7
 Thr  Trp  Tyr  Val
-0.4  1.6  0.5  0.7
```

9.4 Pattern matching

In pattern matching, we generally search a long sequence for a particular subsequence pattern while possibly allowing for a specified number of mismatches in the subsequence. Possible patterns to search for include stop codons `UAG`, `UGA` `UAA` in RNA, protein-DNA or protein-RNA binding sites, and recognition sequences for restriction enzymes (e.g. Roberts, et al., 2007).

Example 1: Pattern matching. In the sequence with NCBI accession number "AF517525.CCND3", we will first search for the pattern "cccggg" with zero mismatches. Then we will repeat the search while allowing for a single mismatch. We use the function `c2s()` to convert a sequence of characters into a single string:

```
> library(seqinr)
> choosebank("genbank")
> ccnd3.human = query("ccnd3.human","sp=homo sapiens AND k=ccnd3@")
> ccnd3.human.DNAseqs <- sapply(ccnd3.human$req, getSequence)
> ccnd3.human.DNAseq1 <- c2s(ccnd3.human.DNAseqs[[1]])
> ccnd3.human.DNAseq1
[1] "atggagctgctgtgttgcgaaggcacccggcacgcgccccgggccgggccggacccgcgg"...
> subseq <- "cccggg"
> countPattern(subseq, ccnd3.human.DNAseq1, mismatch = 0)
[1] 2
> matchPattern(subseq, ccnd3.human.DNAseq1, mismatch = 0)
  Views on a 879-letter BString subject
Subject: atggagctgctgtgttgcgaaggcacccggcacg...actcctacagatgtcacag
Views:
    start end width
[1]    38  43     6 [cccggg]
[2]   809 814     6 [cccggg]
> matchPattern(subseq, ccnd3.human.DNAseq1, mismatch = 1)
  Views on a 879-letter BString subject
Subject: atggagctgctgtgttgcgaaggcacccggcacg...actcctacagatgtcacag
Views:
     start end width
 [1]    26  31     6 [cccggc]
 [2]    37  42     6 [ccccgg]
 [3]    38  43     6 [cccggg]
 [4]    43  48     6 [gccggg]
 [5]    54  59     6 [cccgcg]
 [6]   119 124     6 [cccgcg]
 [7]   236 241     6 [ccctgg]
 [8]   303 308     6 [cctggg]
 [9]   512 517     6 [cccgtg]
[10]   612 617     6 [cacggg]
[11]   642 647     6 [cctggg]
[12]   661 666     6 [tccggg]
[13]   662 667     6 [ccgggg]
[14]   808 813     6 [ccccgg]
[15]   809 814     6 [cccggg]
[16]   810 815     6 [ccgggg]
```

Note that the number of counted patterns while allowing a mismatch is much larger.

9.5 Pairwise alignments

Among the basic questions about a pair of genes or proteins is to what extent they are similar. To answer this question, the two sequences are aligned in a certain manner and then a similarity score is computed. First, in order to understand sequence alignment it is fundamental to have some idea about sequence recursion.

Example 1: Sequence Recursion. The principle of sequence recursion is to be able to generate a sequence by defining any position in the sequence as a function of the previous positions before it. For a simple example, suppose that the first element is one, $x_1 = 1$, and that the sequence is defined by:

$$x_i = x_{i-1} + 1$$

Then we obtain $x_1 = 1$, $x_2 = 2$, $x_3 = 3$, etc, so that the sequence becomes $1, 2, 3, \cdots$.

Another manner to define a sequence is by multiplying the previous value by a constant. For example, let $x_i = 2x_{i-1}$ with $x_1 = 1$. Then the values of the sequence are $x_1 = 1$, $x_2 = 2$, $x_3 = 4$, $x_3 = 8$, etc. Also, we see that in fact $x_n = 2^n$. Thus, a value of the sequence can be computed without actually computing all previous elements.

Another example is $x_i = 2x_{i-1} - 10$, with $x_1 = 1$. In order to compute the value x_{10} we can use R as follows:

```
> x<-double(); x[1]<-1
> for (i in 2:10) {x[i]<- 2*x[i-1]-10}
> x[10]
[1] -4598
```

Example 2: Needleman-Wunsch global alignment. Now, suppose we want to compute an alignment score for two small DNA sequences GAATTC and GATTA (Durbin et. al., 1998, p.18). First, we agree that a match between two letters should have the score +2 and a mismatch the score -1. A gap at a certain position of the sequences should be penalized by subtracting a score $d = 2$. A possible alignment is $\begin{smallmatrix} G\,A\,A\,T\,T\,C \\ G\,A\,T\,T-A \end{smallmatrix}$, where the minus sign indicates a gap. From left to right, the alignment consists of a match, match, mismatch, match, gap, mismatch, respectively, so that the score is $2 + 2 - 1 + 2 - 2 - 1 = 2$. Now an important question is whether this alignment is optimal in the sense that the score is maximal. The answer is: No! To see this, consider the alignment $\begin{smallmatrix} G\,A\,A\,T\,T\,C \\ G\,A-T\,T\,A \end{smallmatrix}$. From left to right, we have a match, match, gap, match, match, mismatch, respectively, so that the score is $2+2-2+2+2-1=5$. The new score is better, but we still do not know whether the new alignment is optimal.

In order to ascertain that the alignment is optimal, we have to build an alignment score matrix $F(i,j)$. To do so, it's convenient to start with building the (mis)match score matrix $s(i,j)$. Its (i,j)th element $s(i,j)$ has the value 2 in case of a match and the value -1 is case of a mismatch. Note that for each step we can choose between a gap, a match, or a mismatch. Building up the matrix $F(i,j)$ recursively means that we define its elements on the basis of the values of its preceding elements. That is, given the values of the previous elements $F(i-1,j-1)$, $F(i-1,j)$, and $F(i,j-1)$, we will be able to find the best consecutive value for $F(i,j)$. In particular, in the case of a match or a mismatch, we take $F(i,j)=F(i-1,j-1)+s(x_i,y_j)$ and in the case of a gap we take $F(i,j)=F(i-1,j)-d$ or $F(i,j)=F(i,j-1)-d$. The famous Needleman-Wunsch alignment algorithm consists of taking the maximum out of these possibilities at each step (e.g, Durbin et. al., 1998, p.21). Their algorithm can be summarized, as follows:

$$F(i,j)=\max\begin{cases} F(i-1,j-1)+s(i,j) \\ F(i-1,j)-d \\ F(i,j-1)-d \end{cases}$$

However, note that this recursion will not yet work because we have not defined any initial values. To solve this, we will start with $F(0,0)=0$ and due to the gap penalties we take $F(i,0)=-id$ for the entries in the first column and $F(0,j)=-jd$ for the entries in the first row. Then, the final score $F(n,m)$ is the optimal score and the values in the matrix $F(i,j)$ indicate the optimal path. This recursive scheme is often called a "dynamic programming algorithm" and the F matrix is called a "dynamic programming matrix".

Example 3: Needleman-Wunsch with a mismatch penalty. Consider again the DNA sequences GAATTC, GATTA, the score +2 for a match, the score -1 for a mismatch, and the gap penalty $d=2$. We first construct the score matrix $s(i,j)$ and call it `s`. For this we use the string-to-character function `s2c()`, a "`for loop`", and an "`if else`" statement:

```
> library(seqinr)
> x <- s2c("GAATTC"); y <- s2c("GATTA"); d <- 2
>
> # create nucleotide substitution matrix
> s <- matrix(data=NA,nrow=length(y),ncol=length(x))
> for (i in 1:(nrow(s))) {
+   for (j in 1:(ncol(s))) {
+     if (y[i]==x[j]) {
+       s[i,j]<- 2 # match
+     }
+     else {
+       s[i,j]<- -1 # mismatch
+     }
+   }
+ }
> rownames(s) <- c(y); colnames(s) <- c(x)
> s
   G  A  A  T  T  C
G  2 -1 -1 -1 -1 -1
A -1  2  2 -1 -1 -1
T -1 -1 -1  2  2 -1
T -1 -1 -1  2  2 -1
A -1  2  2 -1 -1 -1
```

Now we will define the `needleman.wunsch()` function to perform the optimal global alignment of two sequences. To construct the score matrix `F`, we initialize the first row and column of the matrix `F(i,j)` using the `seq()` function. Similarly, to construct the traceback matrix `G`, we initialize the first row and column of the matrix `G(i,j)` using the `rep()` function. Then we use the `max()` function in a "`nested for loop`" to calculate all the other entries in the F and G matricies. The we traverse the traceback matrix `G` backwards from the lower righthand corner to the upper lefthand corner in order to construct the optimal alignment.

```
> needleman.wunsch = function(x, y, sub, d) {
+   # create the dynamic programming matrices F and G
+   F <- matrix(data=NA,nrow=(length(y)+1),ncol=(length(x)+1))
+   G <- matrix(data=NA,nrow=(length(y)+1),ncol=(length(x)+1))
+
+   rownames(F) <- c("",y); colnames(F) <- c("",x)
+   rownames(G) <- c("",y); colnames(G) <- c("",x)
+
+   F[,1] <- -seq(0,length(y)*d,d); F[1,] <- -seq(0,length(x)*d,d)
+   G[,1] <- rep("u",length(y)+1); G[1,] <- rep("l",length(x)+1)
+   G[1,1] = " ";
+
+   for (i in 2:(nrow(F)))
+     for (j in 2:(ncol(F)))
+     {
+         diagonal = F[i-1,j-1]+sub[i-1,j-1]
+         up = F[i-1,j]-d
+         left = F[i,j-1]-d
+         max = max(diagonal, up, left)
+
+         F[i,j] <- max
+
+         if (diagonal == max) {G[i,j] = "d"}
+         else if (up == max) {G[i,j] = "u"}
+         else {G[i,j] = "l"}
+     }
+
+   # Now perform traceback and alignment
+   i=nrow(G); j=ncol(G);
+   max = max(length(x)+length(y)); ii=max; A1=vector(,max); A2=vector(,max);
+
+   repeat{
+       if (G[i,j] == "d") {G[i,j] = "D"; A1[ii] = x[j-1]; A2[ii] = y[i-1]; i=i-1; j=j-1}
+       else if (G[i,j] == "u") {G[i,j] = "U"; A1[ii] = "-"; A2[ii] = y[i-1]; i=i-1}
+       else {G[i,j] = "L"; A1[ii] = x[j-1]; A2[ii] = "-"; j=j-1}
+       ii=ii-1
+       if ((i==1) & (j==1)){
+           break
+       }
+   }
+   return(list(ScoreMatrix=F, TracebackMatrix=G, Alignment=rbind(A1,A2)[,(ii+1):max],
    ↪ Score=F[nrow(F),ncol(F)]))
+ }
>
> dpMatrix = needleman.wunsch(x,y,s,d)a
> dpMatrix
$ScoreMatrix
       G  A  A  T   T   C
    0 -2 -4 -6 -8 -10 -12
G  -2  2  0 -2 -4  -6  -8
A  -4  0  4  2  0  -2  -4
T  -6 -2  2  3  4   2   0
T  -8 -4  0  1  5   6   4
A -10 -6 -2  2  3   4   5

$TracebackMatrix
      G   A   A   T   T   C
  " " "l" "l" "l" "l" "l" "l"
G "u" "D" "L" "l" "l" "l" "l"
A "u" "u" "d" "D" "l" "l" "l"
T "u" "u" "u" "d" "D" "d" "l"
T "u" "u" "u" "d" "d" "D" "l"
A "u" "u" "d" "d" "u" "d" "D"

$Alignment
   [,1] [,2] [,3] [,4] [,5] [,6]
A1 "G"  "A"  "A"  "T"  "T"  "C"
A2 "G"  "-"  "A"  "T"  "T"  "A"

$Score
[1] 5
```

By looking at the lower-right-corner score in the F matrix, we see that the optimal score is indeed 5. The tracebackMatrix allows us to construct the optimal alignment by tracing backwards from the lower righthand corner, where D indicates trace backwards diagonally, L indicates trace backwards to the left, and U indicates trace backwards upward.

Compared to DNA sequences, optimal alignment for pairs of amino acid sequences are often considered to be more relevant since the chemical properties of the amino acids are more closely related to the biological functions of the proteins. For this purpose, we may modify the previous scheme by changing the gap penalty d and the (mis)match scores $s(i,j)$. In particular, we shall use the gap penalty $d = 8$ and the (mis)match scores from the BLOSUM50 matrix.

Table 9.2: BLOSUM50 matrix.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	5	-2	-1	-2	-1	-1	-1	0	-2	-1	-2	-1	-1	-3	-1	1	0	-3	-2	0
R	-2	7	-1	-2	-4	1	0	-3	0	-4	-3	3	-2	-3	-3	-1	-1	-3	-1	-3
N	-1	-1	7	2	-2	0	0	0	1	-3	-4	0	-2	-4	-2	1	0	-4	-2	-3
D	-2	-2	2	8	-4	0	2	-1	-1	-4	-4	-1	-4	-5	-1	0	-1	-5	-3	-4
C	-1	-4	-2	-4	13	-3	-3	-3	-3	-2	-2	-3	-2	-2	-4	-1	-1	-5	-3	-1
Q	-1	1	0	0	-3	7	2	-2	1	-3	-2	2	0	-4	-1	0	-1	-1	-1	-3
E	-1	0	0	2	-3	2	6	-3	0	-4	-3	1	-2	-3	-1	-1	-1	-3	-2	-3
G	0	-3	0	-1	-3	-2	-3	8	-2	-4	-4	-2	-3	-4	-2	0	-2	-3	-3	-4
H	-2	0	1	-1	-3	1	0	-2	10	-4	-3	0	-1	-1	-2	-1	-2	-3	2	-4
I	-1	-4	-3	-4	-2	-3	-4	-4	-4	5	2	-3	2	0	-3	-3	-1	-3	-1	4
L	-2	-3	-4	-4	-2	-2	-3	-4	-3	2	5	-3	3	1	-4	-3	-1	-2	-1	1
K	-1	3	0	-1	-3	2	1	-2	0	-3	-3	6	-2	-4	-1	0	-1	-3	-2	-3
M	-1	-2	-2	-4	-2	0	-2	-3	-1	2	3	-2	7	0	-3	-2	-1	-1	0	1
F	-3	-3	-4	-5	-2	-4	-3	-4	-1	0	1	-4	0	8	-4	-3	-2	1	4	-1
P	-1	-3	-2	-1	-4	-1	-1	-2	-2	-3	-4	-1	-3	-4	10	-1	-1	-4	-3	-3
S	1	-1	1	0	-1	0	-1	0	-1	-3	-3	0	-2	-3	-1	5	2	-4	-2	-2
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	2	5	-3	-2	0
W	-3	-3	-4	-5	-5	-1	-3	-3	-3	-3	-2	-3	-1	1	-4	-4	-3	15	2	-3
Y	-2	-1	-2	-3	-3	-1	-2	-3	2	-1	-1	-2	0	4	-3	-2	-2	2	8	-1
V	0	-3	-3	-4	-1	-3	-3	-4	-4	4	1	-3	1	-1	-3	-2	0	-3	-1	5

Example 4: Needleman-Wunsch with BLOSUM50. For the two sequences "PAWHEAE" and "HEAGAWGHEE" (see, Durbin et. al., 1998, p.21), we seek the Needleman-Wunsch optimal alignment score, using the BLOSUM50 (mis)match score matrix and gap penalty $d = 8$. You can either directly read a BLOSUM matrix from NCBI:

```
>  file <- "ftp://ftp.ncbi.nih.gov/blast/matrices/BLOSUM50"
>  BLOSUM50 <- as.matrix(read.table(file, check.names=FALSE))
```

or load a BLOSUM matrix from the `Biostrings` package.

```
> library(seqinr);library(Biostrings);data(BLOSUM50)
> x <- s2c("HEAGAWGHEE"); y <- s2c("PAWHEAE"); s <- BLOSUM50[y,x]; d <- 8
> dpMatrix = needleman.wunsch(x,y,s,d)
> dpMatrix
$ScoreMatrix
        H   E   A   G   A   W   G   H   E   E
    0  -8 -16 -24 -32 -40 -48 -56 -64 -72 -80
P  -8  -2  -9 -17 -25 -33 -41 -49 -57 -65 -73
A -16 -10  -3  -4 -12 -20 -28 -36 -44 -52 -60
```

```
W -24 -18 -11  -6  -7 -15  -5 -13 -21 -29 -37
H -32 -14 -18 -13  -8  -9 -13  -7  -3 -11 -19
E -40 -22  -8 -16 -16  -9 -12 -15  -7   3  -5
A -48 -30 -16  -3 -11 -11 -12 -12 -15  -5   2
E -56 -38 -24 -11  -6 -12 -14 -15 -12  -9   1

$TracebackMatrix
      H   E   A   G   A   W   G   H   E   E
  " " "L" "L" "l" "l" "l" "l" "l" "l" "l" "l"
P "u" "d" "d" "D" "L" "d" "l" "l" "l" "d" "d"
A "u" "d" "d" "d" "l" "D" "l" "l" "l" "l" "l"
W "u" "u" "u" "d" "d" "d" "D" "L" "l" "l" "l"
H "u" "d" "d" "d" "d" "d" "u" "d" "D" "l" "l"
E "u" "u" "d" "l" "d" "d" "d" "u" "d" "D" "d"
A "u" "u" "u" "d" "l" "d" "d" "d" "u" "U" "d"
E "u" "u" "d" "u" "d" "d" "d" "d" "d" "d" "D"

$Alignment
   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
A1 "H"  "E"  "A"  "G"  "A"  "W"  "G"  "H"  "E"  "-"   "E"
A2 "-"  "-"  "P"  "-"  "A"  "W"  "-"  "H"  "E"  "A"   "E"

$Score
[1] 1
```

Hence, from the lower-right corner of the `ScoreMatrix` we observe that the optimal score equals 1.

Example 5: Needleman-Wunsch with pairwiseAlignment(). We may also conveniently use the `pairwiseAlignment()` function from the `Biostrings` package to find the optimal Needleman-Wunsch aligment score for the sequences "PAWHEAE" and "HEAGAWGHEE" (see, Durbin et. al., 1998, p.21).

```
> library(Biostrings);data(BLOSUM50)
> pairwiseAlignment(AAString("PAWHEAE"), AAString("HEAGAWGHEE"),
+   substitutionMatrix = "BLOSUM50",gapOpening = 0, gapExtension = -8,
+   scoreOnly = FALSE)
Global Pairwise Alignment
1:  --P-AW-HEAE
2:  HEAGAWGHE-E
Score:  1
```

Hence, we obtain the optimal score 1 as well as a representation of the optimal alignment.

An important question is whether the obtained optimal score 1 can be interpreted as being “large” or not. One technique to answer this question is to compare the obtained score with the alignment scores of random sequences. That is, we can compute the probability of alignment scores larger than 1 using random sequences of the same size.

Example 6: Aligning random sequences. To illustrate how the probability of alignment scores larger than 1 can be computed, we sample randomly from the names of the 20 amino acids, seven times for y and 10 times for x, and then compute the maximum alignment score for the two random sequences. This is repeated 1000 times and the probability of optimal alignment scores greater than 1 is estimated by the observed frequency:

```
> library(seqinr);library(Biostrings);data(BLOSUM50)
> allRandomScores <- double()
> for (i in 1:1000) {
+  x <- c2s(sample(rownames(BLOSUM50),7, replace=TRUE))
+  y <- c2s(sample(rownames(BLOSUM50),10, replace=TRUE))
+  allRandomScores[i] <- pairwiseAlignment(AAString(x), AAString(y),
+  substitutionMatrix = "BLOSUM50",gapOpening = 0, gapExtension = -8,
+  scoreOnly = TRUE)
+ }
> sum(allRandomScores>1)/1000
[1] 0.003
```

With the option `scoreOnly = TRUE`, the optimal score is written to the vector `allRandomScores`. The probability of observing scores larger than 1 equals 0.003 and therefore our optimal alignment is much better than expected from randomly constructed sequences.

Example 6: Sliding window of Needleman-Wunsch scores. We may also program a sliding window such that for each window a Needleman-Wunsch alignment score is computed. Then the maximum score can be found and the corresponding region of best alignment identified:

```
> choosebank("genbank"); library(seqinr)
> ccnd3.human = query("ccnd3.human","sp=homo sapiens AND k=ccnd3@")
> ccnd3.human.DNAseqs <- sapply(ccnd3.human$req, getSequence)
> ccnd3.human.AAseqs <- sapply(ccnd3.human.DNAseqs, getTrans)
> x <- c2s(ccnd3.human.AAseqs[[1]])
> y <- c2s(ccnd3.human.AAseqs[[1]][50:70])
> nwscore <- double() ; n <- length(ccnd3.human.AAseqs[[1]])
> for (i in 1:(n-21))
+   nwscore[i] <-
+     pairwiseAlignment(AAString(c2s(ccnd3.human.AAseqs[[1]][i:(i+20)])),
+     AAString(y),substitutionMatrix = "BLOSUM50",gapOpening = 0,
+     gapExtension = -8, scoreOnly = TRUE)
> pairwiseAlignment(AAString(y), AAString(y),
+   substitutionMatrix = "BLOSUM50", gapOpening = 0, gapExtension = -8,
+   scoreOnly = TRUE)
[1] 152
> max(nwscore)
[1] 152
> which.max(nwscore)
```

```
[1] 50
```

Note that the maximum occurs when the subsequences are identical. The value of the maximum score for all the windows is 152 which occurs at window-start-position 50.

9.6 Overview and concluding remarks

In this chapter, we illustrate how the query tools in the `seqinr` library can be used to download sequences, translate them, and compute relevant quantities such as the isoelectric point and hydropathy score. Furthermore, we illustrate how to search sequences for patterns and how optimal pairwise alignments can be programmed. Further applications are given by the exercises below.

Also, the `Biostrings` package contains the various BLOSUM and PAM matrices for optimal alignment, as well as facilities to find palindromes, and to read and write data in FASTA format (`readFASTA()` and `writeFASTA()`).

9.7 Exercises

1. **Writing to a FASTA file.** Query all the human Zyxin genes from Genbank and write the first sequence to a file in FASTA format.

2. **Dotplot of sequences.** Use the function `dotPlot()` of the `seqinr` package and `par(mfrow=c(1,2))` to produce adjacent plots:

   (a) Construct two random sequences of size 100 and plot the first against the second and the first against the first.

   (b) Construct a plot of the: (1) first against the first, and (2) the first against the first in reverse order.

   (c) Download the sequences related to the species homo sapiens and a gene name like "CCND3". Construct a dotplot of the most similar and the least similar sequences. Report your observations.

3. **Smith-Waterman Local alignment.** The Smith-Waterman algorithm seeks the maximum local alignment between *sub*sequences of sequences. Their algorithm can be summarized (Durbin et al., 2005,

p.22) as follows:

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + s(i,j) \\ F(i-1,j) - d \\ F(i,j-1) - d \end{cases}$$

The algorithm allows the score zero if the others have negative values. The idea is that the maximum alignment can occur anywhere in the matrix and that the optimal alignment is defined as the maximum over the whole matrix. Program the Smith-Waterman algorithm and find the optimal local alignment for the sequences "PAWHEAE" and "HEAGAWGHEE".

4. **Probability of a local alignment score.** Sample x and y randomly from the names of the 20 amino acids, seven for y and 10 for x. Repeat this 1000 times and compute the `optimal local alignment` score and use it to evaluate the significance of the previously obtained optimal score.

5. **Prochlorococcus marinus.** Each of three strains of P. marinus is exposed to different intensities of UV radiation because they live in different depths in water. The MIT 9313 strain lives at depth 135 m, SS120 at 120 m, and MED4 at 5 m. The latter strain is considered to be high-light-adapted. The residual intensities of 260-nm UVb irradiation corresponding to the given depths is 0.00007%, 0.0002% and 70%, respectively. It is hypothesized that the GC content in the DNA depends on the amount of radiation each strain is exposed to. The accession numbers in Genbank for the three strains are AE017126, BX548174, and BX548175, respectively.

   (a) Use the operator OR together with the accession numbers to download the sequences of the bacteria strains.

   (b) Compute the GC fraction of each of the sequences.

   (c) Is there a relation between UVb radiation and GC fraction?

   (d) Formulate a relevant hypothesis and test it.

6. **Sequence identity.** Download the sequences "AF517525.CCND3" and "AL160163.CCND3". Hint: These are the first two from the query "ccnd3" within homo sapiens.

(a) Compute the length of the sequences.

(b) Translate the DNA sequences into amino acids and compare their frequencies.

(c) Are they equal or, if not, in what position do they differ?

7. **Conserved regions.** At `http://blocks.fhcrc.org` there are blocks of highly conserved regions for proteins in PROSITE. Find `PR00851A` which contains blocks of protein related to a human gene responsible for DNA-repair defect xeroderma pigmentosum (sensitivity to ultraviolet light). Perform a pairwise alignment with these subsequences and report the ones most and least similar. Use the BLOSUM50 amino acid substitution matrix.

8. **Plot of CG content in C. elegans DNA.**

   (a) Produce a plot of the GC content of chromosome I of C. elegans (Celegans.UCSC.ce2) within sliding windows of 100 nucleotides using the first 10,000 nucleotides.

   (b) A DNA binding site for the enzyme EcoRV is the sequence GATATC. How many exact matches are observed along Chromosome I of C. elegans? How many do you expect by chance?

9. **Codon usage.** Go to the `seqinr` help page for the function `dotchart.uco()`.

   (a) Redo the example found on the help page and briefly describe `dotchart.uco()` usage.

   (b) Use the query language in `seqinr` to find all the human CCND genes and then use the `dotchart.uco()` function to plot their codon usage.