Sequence Alignment and Cell Line Names

Kevin R. Coombes

April 8, 2025

Contents

1 Introduction

A problem that frequently plagues statistical analysts who are trying to combine data from two or more sources is that sample identifiers are entered inconsistently in the two data sets. This problem appears to be particularly prevalent in the world of cell line research, where no standard exists to define the names. Most cell line names can be broken down into three parts: an alphabetic prefix, a numeric identifier, and an optional alphanumeric suffix. Common problems include both the introduction of (apparently random) punctuation between the parts and abbreviation of the prefix (presumably under the belief that, within the context of the experiment, everyone will know what the abbreviation stands for). We recognized that the problem of matching cell line names from two experiments was related to the problem of aligning biological sequence data. As a result, we implemented a straightforward version of the Needleman-Wunsch global alignment algorithm that can be applied to this problem. This vignette explains how to use the *NameNeedle* package to match cell line names.

2 Getting Started

We start by loading the package into the current R session.

```
> library(NameNeedle)
```

3 Aligning Two Character Strings

The basic function is called `needles` and implements the Needleman-Wunsch algorithm for aligning two text strings. The simplest use is to just supply the text strings directly:

```
> needles("hcc-123", "hcc1243")

$score
[1] 4

$align1
[1] "hcc-12*3"

$align2
[1] "hcc*1243"

$sm
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    0   -1   -2   -3   -4   -5   -6   -7
[2,]   -1    1    0   -1   -2   -3   -4   -5
[3,]   -2    0    2    1    0   -1   -2   -3
[4,]   -3   -1    1    3    2    1    0   -1
[5,]   -4   -2    0    2    2    3    2    1
[6,]   -5   -3   -1    1    1    2    4    3
[7,]   -6   -4   -2    0    0    1    3    3
[8,]   -7   -5   -3   -1   -1    0    2    4

$dm
     [,1]   [,2]       [,3]       [,4]       [,5]       [,6]       [,7]
[1,] "none" "left"     "left"     "left"     "left"     "left"     "left"
[2,] "up"   "diagonal" "left"     "left"     "left"     "left"     "left"
[3,] "up"   "up"       "diagonal" "diagonal" "left"     "left"     "left"
[4,] "up"   "up"       "diagonal" "diagonal" "left"     "left"     "left"
[5,] "up"   "up"       "up"       "up"       "diagonal" "diagonal" "left"
[6,] "up"   "up"       "up"       "up"       "diagonal" "up"       "diagonal"
[7,] "up"   "up"       "up"       "up"       "diagonal" "up"       "up"
[8,] "up"   "up"       "up"       "up"       "diagonal" "up"       "up"
     [,8]
[1,] "left"
[2,] "left"
[3,] "left"
[4,] "left"
[5,] "left"
```

```
[6,] "left"
[7,] "diagonal"
[8,] "diagonal"
```

The return value includes the optimal alignments of the two strings, a score, the "score matrix" (`sm`) and the "backtrace matrix" (`dm`). All of these items depend on a set of parameters, which can be supplied to the `needles` function as a list. The default value is

```
> defaultNeedleParams

$MATCH
[1] 1

$MISMATCH
[1] -1

$GAP
[1] -1

$GAPCHAR
[1] "*"
```

Notice that the `GAPCHAR` component specifies the character to use in the alignment strings to indicate that the best alignment involves leaving a gap in one string rather than matching the characters directly in the other string. The other three parameters determine the rewards (for an exact match) or penalties (for mismatches or gaps) that contribute to the score.

A simple way to alter the parameters is to start with the default values and modify the entries:

```
> myParams <- defaultNeedleParams
> myParams$MISMATCH <- -2
> myParams$MATCH <- 2
```

We can re-run the algorithm using our own parameter list.

```
> needles("hcc-123", "hcc1243", myParams)

$score
[1] 10

$align1
[1] "hcc-12*3"

$align2
[1] "hcc*1243"
```

```
$sm
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    0   -1   -2   -3   -4   -5   -6   -7
[2,]   -1    2    1    0   -1   -2   -3   -4
[3,]   -2    1    4    3    2    1    0   -1
[4,]   -3    0    3    6    5    4    3    2
[5,]   -4   -1    2    5    4    7    6    5
[6,]   -5   -2    1    4    3    6    9    8
[7,]   -6   -3    0    3    2    5    8    7
[8,]   -7   -4   -1    2    1    4    7   10

$dm
     [,1]   [,2]       [,3]       [,4]       [,5]       [,6]       [,7]
[1,] "none" "left"     "left"     "left"     "left"     "left"     "left"
[2,] "up"   "diagonal" "left"     "left"     "left"     "left"     "left"
[3,] "up"   "up"       "diagonal" "diagonal" "left"     "left"     "left"
[4,] "up"   "up"       "diagonal" "diagonal" "left"     "left"     "left"
[5,] "up"   "up"       "up"       "up"       "diagonal" "diagonal" "left"
[6,] "up"   "up"       "up"       "up"       "diagonal" "up"       "diagonal"
[7,] "up"   "up"       "up"       "up"       "diagonal" "up"       "up"
[8,] "up"   "up"       "up"       "up"       "diagonal" "up"       "up"
     [,8]
[1,] "left"
[2,] "left"
[3,] "left"
[4,] "left"
[5,] "left"
[6,] "left"
[7,] "diagonal"
[8,] "diagonal"
```

Note that, in this case, the alignment does not change, but the score and the score matrix both change to reflect the altered parameters. In more complex cases, the actual alignments can change depending on the set of parameters. We find that the values in `myParams` work well for matching cell line names.

4 Cell Line Names

The *NameNeedle* library includes a sample data set consisting of the actual cell line names, as they were presented to us, from three related experiments. We use the `data` command to load these names into the current R session.

```
> data(cellLineNames)
> ls()

[1] "illuNames" "illuType"  "myParams"  "rppaNames" "sf2Names"
```

Now we review the cell line names.

```
> class(sf2Names)

[1] "character"

> length(sf2Names)

[1] 129

> sf2Names[1:10]

 [1] "UMSCC17A"  "UMSCC14A"  "UMSCC47"   "PCI-15A"   "PCI-13"    "UMSCC14B"
 [7] "PCI-15B"   "SN-2"      "JHU-011"   "MDA1386TU"

> class(rppaNames)

[1] "character"

> length(rppaNames)

[1] 260

> rppaNames[1:10]

 [1] "344SQ" "383B"  "393LN" "393P"  "3KT"   "584"   "A431"  "A549"  "C39"
[10] "C42"

> class(illuNames)

[1] "character"

> length(illuNames)

[1] 105

> summary(illuType)

HNSCC  Lung
   33    72

> illuNames[1:10]

 [1] "UMSCC 10B" "OSC19 LN2" "TU159"     "TU686"     "UMSCC 14B" "A431"
 [7] "UMSCC 11A" "TMAR-B"    "C39"       "DM-14"
```

4.1 Matching One Cell Line Name

The `needles` function works on one character string at a time. (If you supply a character vector of length greater than one, it silently ignores everything except the first entry). In order to find the best match for one name in a list of possibilities, we use the `needleScores` function. For example, suppose we want to find the best match for the following name

```
> probeName <- sf2Names[6]
> probeName

[1] "UMSCC14B"
```

in the character vector `illuNames`. Then we simply write

```
> scores <- needleScores(probeName, illuNames, myParams)
> summary(scores)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-13.000  -9.000  -8.000  -6.352  -3.000  15.000
```

We see that the highest score is 11. We can use the usual R tools to figure out which character string gives the highest score (and thus the best match).

```
> w <- which(scores==max(scores))
> illuNames[w]

[1] "UMSCC 14B"
```

We note that the match differs from the probe name by the insertion of a space character between the alphabetic prefix and the numerical identifier in the name.

4.2 Matching All Cell Line Names

More generally, we would like to match two complete lists of names. For example, we might want to match the names in `sf2names` with the names in `rppaNames`. There is no special function to perform this task. Instead, we can simply run through a loop.

```
> go <- proc.time()
> matchscore <- matchcode <- rep(NA, length(sf2Names))
> for (j in 1:length(sf2Names)) {
+   scores <- needleScores(sf2Names[j], rppaNames, myParams)
+   matchcode[j] <- paste(which(scores==max(scores)), collapse=',')
+   matchscore[j] <- max(scores)
+ }
> used <- proc.time() - go
```

Note, however, that we must allow for the possibility that multiple names in `rppaNames` will provide the best match (highest score) for any given name in `sf2Names`. We have saved the indices of all the best matches in the `matchcode` variable, as a comma-separated character string. The next loop expands those indices to the actual names, which are stored as semicolon-separated character strings.

```
> rppaMatch <- sapply(matchcode, function(x) {
+   y <- as.numeric(strsplit(x, ',')[[1]])
+   paste(rppaNames[y], collapse="; ")
+ })
```

For example, we have

```
> i <- 116
> sf2Names[i]

[1] "HCC-2998"

> rppaMatch[i]

             84,91
"HCC2279; HCC2935"
```

In this case, sample "*HCC-2998*" was used in the SF2 study but not in the RPPA study, and there are two cell lines in the RPPA study that give equally good (although incorrect) matches. If we want to check the actual alignments, we can again use the basic `needles` function.

```
> x <- needles("HCC-2998", "HCC2279", myParams)
> x$align1

[1] "HCC-2998"

> x$align2

[1] "HCC2279*"
```

For completeness, we also match `sf2names` to `illuNames`.

```
> go <- proc.time()
> imatchscore <- imatchcode <- rep(NA, length(sf2Names))
> for (j in 1:length(sf2Names)) {
+   scores <- needleScores(sf2Names[j], illuNames, myParams)
+   imatchcode[j] <- paste(which(scores==max(scores)), collapse=',')
+   imatchscore[j] <- max(scores)
+ }
> illuMatch <- sapply(imatchcode, function(x) {
+   y <- as.numeric(strsplit(x, ',')[[1]])
+   paste(illuNames[y], collapse="; ")
+ })
> iused <- proc.time() - go
> used
```

```
   user  system elapsed
   2.09    0.00    2.10

> iused

   user  system elapsed
   0.78    0.01    0.81

> used + iused

   user  system elapsed
   2.87    0.01    2.91
```

We can combine the results into a data frame.

```
> matcher <- data.frame(rppaMatch=rppaMatch, rppaScore=matchscore,
+                       illuMatch=illuMatch, illuScore=imatchscore)#,combined)
> rownames(matcher) <- sf2Names
> matcher[1:10,]

          rppaMatch rppaScore             illuMatch illuScore
UMSCC17A   UMSCC17A        16             UMSCC 11A        11
UMSCC14A   UMSCC14A        16  UMSCC 14B; UMSCC 11A        11
UMSCC47     UMSCC47        14             UMSCC 14B         8
PCI-15A      PCI15A        11                 HCC15         0
PCI-13        PCI13         9 C39; HCC1359; HCC1833        -1
UMSCC14B   UMSCC14B        16             UMSCC 14B        15
PCI-15B      PCI15B        11                 HCC15         0
SN-2            SN2         5             OSC19 LN2        -1
JHU-011      JHU011        11               HCC4011         2
MDA1386TU   MDA1386        12              MDA 1386        11
```

5 Conclusions

A simple implementation of the Needleman-Wunsch global alignment algorithm works reasonably well as a first approximation for matching cell line names from different datasets. We must note, however, that more sophisticated tools are available for aligning biological sequences; many such tools are implemented in the *Biostrings* package from Bioconductor.