Design Group-Specific FISH Probes

Erik S. Wright

May 2, 2019

Contents

1 Introduction

This document describes how to design group-specific FISH probes using the DECIPHER package through the use of the `DesignProbes` function. Fluorescent In-Situ Hybridization (FISH) is an laboratory technique that enables visualizing specific microorganisms in their natural environment using fluorescent microscopy. As a case study, this tutorial focuses on the small-subunit ribosomal RNA (SSU rRNA) from a collection of bacteria found in drinking water. Here we describe how to design probes targeting a group of interest that will not cross-hybridize with other bacteria from the same sample. A similar strategy could be used with any set of aligned sequences that are separated into groups. For example, genus-specific probes designed with this program that target all named bacteria and archaea genera are provided online at `http://DECIPHER.codes/16SOligos.html`.

A database of aligned DNA sequences separated into groups is used as input to the program. First the function `TileSeqs` is used to pre-process the sequences into overlapping tiles, which will serve as the template DNA for probe design. Second, the `DesignProbes` function determines the set of all possible probes that meet certain design constraints, such as the ability to hybridize with the group of interest under specified experimental conditions. Next, the complete set of probes is scored by its potential to cross-hybridize

with sequences from other groups including a comprehensive database of non-target sequences. Finally, the optimal set of dual probes is chosen that could be used in a FISH experiment to identify the microbes of interest.

2 The Objective of FISH Probe Design

The objective of FISH probe design is to balance sensitivity of the probe to the target group of sequences, while simultaneously maximizing specificity to prevent cross-hybridizations with other non-target groups. Here there is a fundamental trade-off, because if a probe is too weak it will not bind to its target, but if too strong it will also bind to non-targets. This balancing act is complicated by the difficulty of accurately predicting binding strength of the probe with a variety of potential target and non-target sequences. To confront this challenge, `DesignProbes` uses a state of the art model of probe hybridization in the presence of the denaturant formamide. This model, implemented in the function `CalculateEfficiencyFISH`, is used in conjunction with reasonable safety factors to minimize the need for experimental optimization.

In cases where the desired level of specificity cannot be achieved, `DesignProbes` predicts the degree of potential cross-hybridization with non-target groups. When a single probe is insufficient, dual probes with distinct colors can be employed to further increase specificity to the target group. `DesignProbes` will choose the optimal combination of dual probes that will minimize cross-hybridization overlap. During the experiment, the combination of the individual probe colors provides additional assurance of a positive identification. For example, if one probe is labeled green (e.g., fluorescein) and the other is labeled red (e.g., Cy3) then the hybridization of both probes at the same point would appear yellow.

3 Getting Started

3.1 Installing OligoArrayAux

The program OligoArrayAux (`http://mfold.rna.albany.edu/?q=DINAMelt/OligoArrayAux`) is used to predict hybridization efficiency and must be installed in a location accessible by the system. For example, the following code should print the installed OligoArrayAux version when executed from the *R* console:

```
> system("hybrid-min -V")

hybrid-min (OligoArrayAux) 3.8
By Nicholas R. Markham and Michael Zuker
Copyright (C) 2006
Rensselaer Polytechnic Institute
Troy, NY 12810-3590 USA
```

3.2 Startup

To get started we need to load the DECIPHER package, which automatically loads several other required packages.

```
> library(DECIPHER)
```

Help for the `DesignProbes` function can be accessed through:

```
> ? DesignProbes
```

If DECIPHER is installed on your system, the code in each example can be obtained via:

```
> browseVignettes("DECIPHER")
```

3.3 Creating a Sequence Database

We begin with a set of aligned 16S sequences representing a variety of bacteria collected from drinking water samples. We wish to design probes targeting only the genus *Sphingopyxis*, which is a common organism detected in drinking water distribution systems. This example uses a GenBank sequence file included as part of the DECIPHER package, but you could follow along with your own GenBank, FASTA, or FASTQ file of aligned sequences. Be sure to change the path names to those on your system by replacing all of the text inside quotes labeled "<<path to ...>>" with the actual path on your system.

```
> # specify the path to your sequence file:
> gb <- "<<path to GenBank file>>"
> # OR find the example sequence file used in this tutorial:
> gb <- system.file("extdata", "Bacteria_175seqs.gen", package="DECIPHER")
```

Next, there are two options for importing the sequences into a database: either save a database file or maintain the database in memory. Here we will build the database in memory because it is a small set of sequences and we do not intend to use the database later:

```
> # specify a path for where to write the sequence database
> dbConn <- "<<path to write sequence database>>"
> # OR create the sequence database in memory
> dbConn <- dbConnect(SQLite(), ":memory:")
> Seqs2DB(gb, "GenBank", dbConn, "Bacteria")
Reading GenBank file chunk 1

175 total sequences in table Seqs.
Time difference of 0.09 secs
```

3.4 Defining Groups

At this point we need to define groups of related sequences in the database we just created. In this case we wish to define phylogenetic groups based on the taxonomic information included in the original GenBank file. The advantage of using a GenBank file is that the taxonomic information is automatically imported from the ORGANISM field into a "rank" column in the sequence database. When importing from a FASTA or FASTQ file the taxonomic information must come from another source, such as parsing the sequence identifiers or by building a tree with the function `IdClusters`. In the case of a GenBank file, the "rank" column can be used by the function `IdentifyByRank` to identify the groups at any taxonomic level (e.g., genus).

```
> ids <- IdentifyByRank(dbConn, level=Inf, add2tbl=TRUE)
Updating column: "identifier"...
Formed 72 distinct groups.
Added to table Seqs: "identifier".
Time difference of 0.06 secs
```

4 Probe Design Steps

4.1 Tiling Sequences

We continue by creating a set of k-mers, known as "tiles", that represent the sequences in each group. Here we must make several decisions that will affect probe design in the future. The defaults are to create tiles of

length 26-27 nucleotides with up to 10 permutations that represent at least 90% of the permutations present in each target site. These parameters will generally fit most probe designs, but it is recommended to read the help file for the `TileSeqs` function to make sure that it is doing what is desired. We will save the resulting tiles back to the database as a new table named "Tiles" in case we wish to access them in the future.

```
> tiles <- TileSeqs(dbConn, add2tbl="Tiles")


  |======================================================================| 100%

Time difference of 572.06 secs
```

4.2 Designing All Possible Probes

Next we wish to design probes targeting the 5 sequences identified as belonging to the genus *Sphingopyxis*. As an example, we will begin by designing probes for every possible target site that meet certain design criteria. By default, we allow up to 4 probe permutations as long as they cover 90% of the permutations present in their target site. Note that the input parameters to `DesignProbes` should be carefully considered in order adequately represent the experimental conditions that will be used. For example, here we limit the target sites to those between alignment positions 120 and 1,450 because this is the region shared by most of the sequences.

```
> probes <- DesignProbes(tiles, identifier="Sphingopyxis",
         start=120, end=1450)


Sphingopyxis (678):
  |======================================================================| 100%

Time difference of 250 secs
```

We can now examine the top-scoring probe with maximal sensitivity and specificity to the target group.

```
> o <- order(probes$score, decreasing=TRUE)
> probes[o[1],]


      identifier start start_aligned permutations
826 Sphingopyxis   859          1019            3
           score                 probe.1
826 -1.43876.... CCGCGATTAGGATGTCAAACGCT
                    probe.2                 probe.3 probe.4
826 CCGCGATCAGGATGTCAAATGCT CCGCGATCAGGATGTCAAACGCT    <NA>
    efficiency.1 efficiency.2 efficiency.3 efficiency.4
826    0.5014489    0.5506237    0.6315190           NA
       FAm.1    FAm.2    FAm.3    FAm.4 coverage.1
826 35.04488 36.57339 39.17166       NA        0.4
    coverage.2 coverage.3 coverage.4
826        0.2        0.4         NA
mismatches
826 unclassified_Sphingomonadaceae (63.2%,0.00kcal/mol,
4.1%;CCGCGATCAGGATGTCAAACGCT/AGCGTTTGACATCCTGATCGCGG)
unclassified_Sphingomonadales (9.1%,6.20kcal/mol,
-17.8%;CCGCGATCAGGATGTCAAATGCT/AGCTTTTGACATCCCGGTCGCGG)
```

The best probe out of 678 candidate target sites has three permutations, with formamide melt points (FAm) between 35% and 39% [FA] (v/v). These three permutations can be represented with a single consensus probe:

```
> ConsensusSequence(DNAStringSet(probes[o[1], "probe"][1:3]))


  A DNAStringSet instance of length 1
    width seq
[1]    23 CCGCGATYAGGATGTCAAAYGCT
```

Based on the column *mismatches*, this probe has two predicted cross-hybridizations with other groups in the sequence set. These groups, unclassified Sphingomonadaceae and unclassified Sphingomonadales, are predicted to have 63.2% and 9.1% hybridization efficiency at equilibrium in the hybridization buffer with 35% [FA] (v/v). The first non-target has exactly the same Gibb's free energy because it is a perfect match to the probe. Note that its formamide melt point (FAm) is 4.1% greater than the probe permutation with the lowest melt point, which is the limiting factor in the experiment. The second probe has 6.2 kcal/mol greater Gibb's free energy and a formamide melt point 17.8% [FA] less than the probe with the target. We can examine the predicted melt curve of the probes with the target and non-target:

```
> FA_range <- 0:70 # [FA] (%, v/v)
> probe <- probes$probe[o[1], 1:3]
> targets <- reverseComplement(DNAStringSet(probe))
> f <- function(FA)
        CalculateEfficiencyFISH(probe, targets,
                temp=46, P=250e-9, ions=1, FA)[, "HybEff"]
> efficiency <- matrix(unlist(lapply(FA_range, f)), ncol=3, byrow=TRUE)
> matplot(FA_range, efficiency, ylim=c(0,1), ylab="Hybridization Efficiency",
        xlab=expression(paste("[Formamide] (%, v/v)", sep="")),
        type="l", lwd=2, col="Blue", main="Formamide Curve", lty=1)
> nontargets <- DNAStringSet(c("AGCGTTTGACATCCTGATCGCGG",
        "AGCTTTTGACATCCCGGTCGCGG"))
> f <- function(FA)
        CalculateEfficiencyFISH(probe[3:2], nontargets,
                temp=46, P=250e-9, ions=1, FA)[, "HybEff"]
> efficiency <- matrix(unlist(lapply(FA_range, f)), ncol=2, byrow=TRUE)
> matlines(FA_range, efficiency, col="Red", lwd=2, lty=3)
> abline(h=0.5, lty=2, lwd=2, col="Orange")
> abline(v=35, lty=2, lwd=2, col="Green")
> legend("topright", legend=c("Targets", "Non-Targets", "50% Efficiency",
        "Experimental [FA]"), col=c("Blue", "Red", "Orange", "Green"),
        lwd=c(2, 2, 2, 2), lty=c(1, 3, 2, 2))
```

4.3 Finding Non-targets in a Comprehensive Reference Database

It is now important to find potential non-targets not included in the sequence set used for design. For this purpose we can either build a new comprehensive reference database, or download one online from http://DECIPHER.cee.wisc.edu/Download.html. The reference database must be unzipped after download. Each database includes a pre-processed set of tiles, generated in the same manner as shown above, that can be found in the table named "Tiles". Be sure to change the path names to those on your system by replacing all of the text inside quotes labeled "<<path to ...>>" with the actual path on your system.

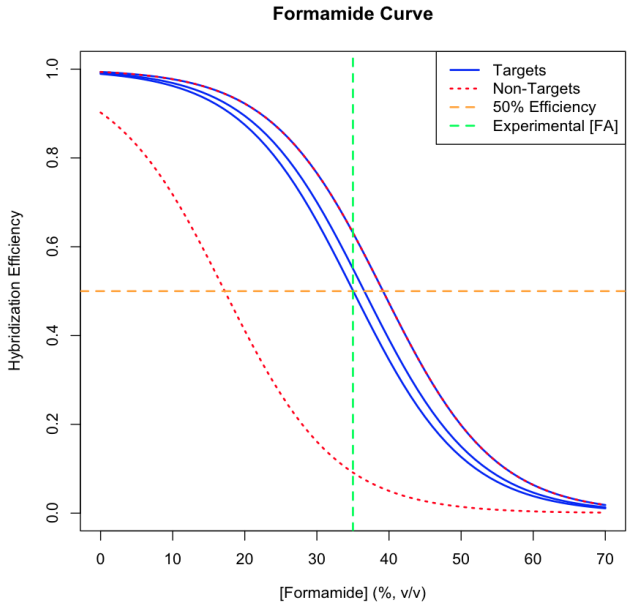

Figure 1: Formamide melt curves for the targets and non-targets

```
> dbConn_ref <- dbConnect(SQLite(), "<<path to reference database>>")
> # select the most common k-mers
> ref_tiles <- dbGetQuery(dbConn_ref,
        "select * from Tiles where groupCoverage > 0.2 and coverage > 0.01")
> dbDisconnect(dbConn_ref)
> ref_tiles$id <- paste("ref", ref_tiles$id, sep="_")
```

Next we can use these tiles to find hits in the comprehensive database of non-targets that may not have been included in the sequence set used for design. This step is unnecessary if a comprehensive sequence set was used during the design steps above. Here we used a reduced set of non-targets during design, so we must search the comprehensive reference database for other non-targets.

This step requires searching for a substantial number of probes in a very large number of non-targets. To accomplish this feat in a reasonable time we will leverage Biostrings' dictionary matching functionality. Nevertheless this step still requires several minutes to complete. Note that this process can be accelerated by eliminating the lowest scoring probes (i.e., `probes <- probes[o[1:20],]`), but this should only be used in cases where the number of non-targets included in the design sequence set was sufficient to narrow down the list of potential probes to reasonable candidates.

```
> seqs <- DNAStringSet(ref_tiles$target_site)
> w <- which(!is.na(t(probes$probe)))
> probes_rc <- reverseComplement(DNAStringSet(t(probes$probe)[w]))
> p <- PDict(probes_rc, tb.start=1, tb.width=5)
> hits1 <- vwhichPDict(p, seqs, max.mismatch=5)
> l <- vapply(hits1, length, integer(1))
> hits1 <- unlist(hits1, use.names=FALSE)
> names(hits1) <- rep(1:length(l), l)
> p <- PDict(probes_rc, tb.end=-1, tb.width=5)
> hits2 <- vwhichPDict(p, seqs, max.mismatch=5)
```

```
> l <- vapply(hits2, length, integer(1))
> hits2 <- unlist(hits2, use.names=FALSE)
> names(hits2) <- rep(1:length(l), l)
> hits <- c(hits1, hits2)
```

Now we can use the thermodynamic model to predict the degree of cross-hybridization between each probe and non-target. This information is used to update the score for every probe, and is stored in the column *mismatches* alongside previous non-targets identified during design. Refer to section 3.2 above for how to easily obtain the code shown below:

```
> Hyb_FA <- 35 # the default Hybridization [FA] (%; v/v)
> count <- 0L
> pBar <- txtProgressBar(style=3)
> for (i in 1:dim(probes)[1]) {
        # for each hit calculate the degree of cross-hybridization
        w <- which(!is.na(probes[i, "probe"]))
        results <- NULL
        for (j in 1:length(w)) {
                count <- count + 1L
                w <- which(hits==count)
                if (length(w) > 0) {
                        ns <- as.integer(unique(names(hits[w])))
                        eff <- CalculateEfficiencyFISH(rep(probes$probe[i, j],
                                        length(ns)),
                                ref_tiles$target_site[ns],
                                46, # temperature
                                250e-9, # [Probe]
                                1, # [NA]
                                Hyb_FA)
                        eff <- cbind(eff,
                                data.frame(id=ref_tiles$id[ns],
                                        probe_rc=toString(probes_rc[count]),
                                        target=ref_tiles$target_site[ns],
                                        dFAm=eff[, "FAm"] - Hyb_FA,
                                        stringsAsFactors=FALSE))
                        results <- rbind(results, eff)
                }
        }

        w <- which(results$dFAm > -20)
        if (length(w) > 0) {
                # only record the strongest cross-hybridization in each group
                results <- results[w,]
                u <- unique(results$id)
                keep <- integer()
                for (j in 1:length(u)) {
                        w <- which(results$id==u[j])
                        keep <- c(keep, w[which.max(results$dFAm[w])])
                }
                results <- results[keep,]

                # append more non-targets to mismatches
```

```
                p <- pairwiseAlignment(results$probe_rc,
                        results$target,
                        type="global-local")
                probes$mismatches[i] <- paste(probes$mismatches[i],
                        paste(results$id,
                                " (", round(100*results$HybEff, 1), "%,",
                                round(results$ddG1, 2), "kcal/mol,",
                                round(results$dFAm, 1), "%;",
                                substring(reverseComplement(DNAStringSet(pattern(p))),
                                        1L),
                                "/", substring(subject(p), 1L), ")",
                                sep="",
                                collapse=" "),
                        sep="")

                # score -= 0.2 + 1.2^dFAm
                probes$score[i] <- probes$score[i] -
                        sum(ifelse(results$dFAm < -20, 0, 0.2 +
                                1.2^ifelse(results$dFAm > 0, 0, results$dFAm)))
        }

        setTxtProgressBar(pBar, i/dim(probes)[1])
}


  |=======================================================================| 100%
```

Finally, we can examine the top scoring single probe after updating the results using the comprehensive database. The new best scoring probe targets a site over 300 nucleotides away from the original best scoring probe. The design set was missing many non-targets that were found by searching the comprehensive database. Note that the target group, *Sphingopyxis*, was present in the comprehensive database and now appears in the set of non-targets (`mismatches`). This has no net impact on the scores however, as all of the candidate probes are effected equally since they are all perfect match to sequences in this group.

```
> # the original best scoring probe after searching the comprehensive database
> probes[o[1],]


      identifier start start_aligned permutations        score
826 Sphingopyxis   859          1019            3 -7.75865....
                    probe.1                 probe.2
826 CCGCGATTAGGATGTCAAACGCT CCGCGATCAGGATGTCAAATGCT
                    probe.3 probe.4 efficiency.1 efficiency.2
826 CCGCGATCAGGATGTCAAACGCT    <NA>    0.5014489    0.5506237
    efficiency.3 efficiency.4    FAm.1    FAm.2    FAm.3    FAm.4
826    0.6315190           NA 35.04488 36.57339 39.17166       NA
    coverage.1 coverage.2 coverage.3 coverage.4
826        0.4        0.2        0.4         NA

826 unclassified_Sphingomonadaceae (63.2%,0.00kcal/mol,4.1%;CCGCGATCAGGATGTCAAACGCT/
AGCGTTTGACATCCTGATCGCGG) unclassified_Sphingomonadales (9.1%,6.20kcal/
mol,-17.8%;CCGCGATCAGGATGTCAAATGCT/AGCTTTTGACATCCCGGTCGCGG) ref_Novosphingobium
```

```
(13.2%,4.68kcal/mol,-14.6%;CGCGATTAGGATGTCAAACGCT/AGCGTTTGACATCCTCATCGCG)
ref_Sphingopyxis (63.2%,0kcal/mol,4.2%;CCGCGATCAGGATGTCAAACGCT/
AGCGTTTGACATCCTGATCGCGG) ref_Zymomonas (63.2%,0kcal/mol,
4.2%;CCGCGATCAGGATGTCAAACGCT/AGCGTTTGACATCCTGATCGCGG) ref_Oceanicaulis (7.5%,
6.25kcal/mol,-19.5%;CCGCGATTAGGATGTCAAACGCT/TGCGCTTGACATCCCTATCGCGG)
ref_Cucumibacter (25.9%,3.11kcal/mol,-8.2%;CCGCGATCAGGATGTCAAATGCT/
AGCTCTTGACATCCTGATCGCGG) ref_Celeribacter (15%,5.64kcal/
mol,-13.5%;CCGCGATCAGGATGTCAAACGCT/AACCCTTGACATCCTGATCGCGG) ref_Jannaschia (15%,
5.64kcal/mol,-13.5%;CCGCGATCAGGATGTCAAACGCT/AACCCTTGACATCCTGATCGCGG)
ref_Maritimibacter (15%,5.64kcal/mol,-13.5%;CCGCGATCAGGATGTCAAACGCT/
AACCCTTGACATCCTGATCGCGG) ref_Pseudoruegeria (15%,5.64kcal/
mol,-13.5%;CCGCGATCAGGATGTCAAACGCT/AACCCTTGACATCCTGATCGCGG) ref_Rhodovulum (15%,
5.64kcal/mol,-13.5%;CCGCGATCAGGATGTCAAACGCT/AACCCTTGACATCCTGATCGCGG) ref_Thioclava
(15%,5.64kcal/mol,-13.5%;CCGCGATCAGGATGTCAAACGCT/AACCCTTGACATCCTGATCGCGG)
ref_Altererythrobacter (24.4%,3.31kcal/mol,-8.8%;CCGCGATCAGGATGTCAAATGCT/
AGCCTTTGACATCCTGGTCGCGG) ref_Croceicoccus (24.4%,3.31kcal/
mol,-8.8%;CCGCGATCAGGATGTCAAATGCT/AGCCTTTGACATCCTGGTCGCGG) ref_Afifella (8.7%,
6.32kcal/mol,-18.2%;CCGCGATCAGGATGTCAAATGCT/AGCTCTTGACATCCCGATCGCGG) ref_Breoghania
(8.7%,6.32kcal/mol,-18.2%;CCGCGATCAGGATGTCAAATGCT/AGCTCTTGACATCCCGATCGCGG)


> # the new best scoring probe after searching the comprehensive database
> o <- order(probes$score, -1*probes$permutations,
         rowSums(as.matrix(probes$coverage[,]), na.rm=TRUE),
         decreasing=TRUE)
> probes[o[1],]


      identifier start start_aligned permutations        score
489 Sphingopyxis   522           679            2 -5.93961....
                      probe.1                   probe.2 probe.3 probe.4
489 CACTGACCTCTCCAAGATTCAAGTT CACTGACCTCTCCAAGATTCTAGTT    <NA>    <NA>
    efficiency.1 efficiency.2 efficiency.3 efficiency.4    FAm.1
489    0.5700456    0.6445774           NA           NA 37.04415
       FAm.2    FAm.3    FAm.4 coverage.1 coverage.2 coverage.3
489 39.31453       NA       NA        0.8        0.2         NA
    coverage.4
489         NA

489 Porphyrobacter (12.3%,6.35kcal/mol,-16.3%;CACTGACCTCTCCAAGATTCTAGTT/
AGCTAGAATCTTGGAGAGGCGAGTG) Blastomonas (9.2%,6.36kcal/
mol,-18.6%;CACTGACCTCTCCAAGATTCAAGTT/GGCTTGAATCTTGGAGAGGCGAGTG) Novosphingobium
(54.1%,1.07kcal/mol,-0.9%;CACTGACCTCTCCAAGATTCTAGTT/AGCTAGAATCTTGGAGAGGTCAGTG)
unclassified_Sphingomonadaceae (12.3%,6.34kcal/mol,-16.3%;CACTGACCTCTCCAAGATTCTAGTT/
CGCTAGAATCTTGGAGAGGCGAGTG) Spirochaeta (12.6%,6.28kcal/
mol,-16.1%;CACTGACCTCTCCAAGATTCTAGTT/GACTAGAATCTTGGAGGGGGAGTTG) ref_Sphingopyxis
(57%,0kcal/mol,2%;CACTGACCTCTCCAAGATTCAAGTT/AACTTGAATCTTGGAGAGGTCAGTG)
ref_Croceicoccus (65.8%,-0.15kcal/mol,4.7%;CACTGACCTCTCCAAGATTCTAGTT/
GACTAGAATCTTGGAGAGGTCAGTG) ref_Novosphingobium (65.8%,-0.15kcal/mol,
4.7%;CACTGACCTCTCCAAGATTCTAGTT/GACTAGAATCTTGGAGAGGTCAGTG) ref_Truepera (14.3%,
5.91kcal/mol,-13%;CACTGACCTCTCCAAGATTCTAGTT/AACTAGACTCTTGGAGAGGTAAGTA)
```

5 Dual Probe Design Steps

5.1 Designing the Optimal Dual Probe Set

Since no single probe was sufficient to completely distinguish all non-targets, we can further increase specificity by using two probes in combination. There are two methods to accomplish the design of dual probes:

1. If using a comprehensive sequence set during design, then we could have simply set the parameter *numProbeSets* to a positive number from its default of zero. In doing so the `DesignProbes` function will search through potential combinations of two probes. The value of *numProbeSets* will determine how many dual probe sets `DesignProbes` will design. For example:

```
> probes <- DesignProbes(tiles, identifier="Sphingopyxis",
         start=120, end=1450,
         numProbeSets=100) # note numProbeSets > 0


Sphingopyxis (678):
  |======================================================================| 100%

Time difference of 270 secs


> dim(probes) # now there are 100 potential probe sets


[1] 100 21
```

2. If searching through non-targets using a comprehensive reference database (as outlined in the sections above) then it is simpler to continue from where we left off above, because we already finished designing all single probes and searching for potential non-targets. In this case, all we need to do is search for the combination of two probes with the minimal mismatch overlap, as described below.

To continue, we first compute the score for using each pair of probes as a dual probe set. The new score for using two probes is based off the minimal cross-hybridization between each pair of probes and every non-target. For example, if probe #1 cross-hybridizes with a non-target, but probe #2 does not, then the non-target is not a threat when the two probes are used in conjunction because the hybridization of both probes is required for a positive identification.

```
> numProbeSets <- 100 # number of probe sets to generate
> s <- ifelse(dim(probes)[1] > numProbeSets, numProbeSets, dim(probes)[1])
> MMs <- strsplit(probes$mismatches[o[1:s]], "mol,", fixed=TRUE)
> ls <- unlist(lapply(MMs, length))
> ls <- ifelse(ls > 0, ls - 1, 0)
> index <- rep(1:length(ls), ls)
> if (length(index) > 0) {
         MMs <- strsplit(unlist(MMs), "%;", fixed=TRUE)
         MMs <- unlist(strsplit(unlist(MMs), ") ", fixed=TRUE))
         effs <- as.numeric(MMs[seq(2, length(MMs), 3)])
         MMs <- MMs[seq(1, length(MMs), 3)]
         MMs <- unlist(strsplit(MMs, " (", fixed=TRUE))
         MMs <- MMs[seq(1, length(MMs), 2)]
```

```
 }
> m <- matrix(0, nrow=s, ncol=s, dimnames=list(o[1:s], o[1:s])) # scores
> n <- matrix(0, nrow=s, ncol=s, dimnames=list(o[1:s], o[1:s])) # counts
> for (i in 1:(s - 1)) {
        w1 <- which(index==i)
        if (length(w1)==0)
                next
        for (j in (i + 1):s) {
                w2 <- which(index==j)
                if (length(w2)==0)
                        next
                overlap_MMs <- match(MMs[w1], MMs[w2])
                w <- which(!is.na(overlap_MMs))
                n[i, j] <- length(w)
                if (length(w) > 0)
                        m[i, j] <- sum(tapply(c(effs[w1[w]], effs[w2[overlap_MMs[w]]]),
                                rep(1:length(w), 2),
                                function(x) return(-1.2^ifelse(min(x) > 0, 0, min(x)))))
        }
 }
```

As expected, the score for using two probes in combination is significantly better than using any single probe alone. This can be seen in Figure 2, where the off-diagonal colors represent the use of two probes, and the diagonal elements represent the use of a single probe. Some combinations of two probes results in poor scores (seen in red/magenta), while others give high scores (blue/green).

```
> # plot the matrix of dual-probe scores
> m <- m + t(m) # make symmetric
> diag(m) <- probes$score[o[1:s]]
> O <- order(o[1:s]) # unsort
> cols <- colorRamp(rainbow(101, start=0, end=0.35, v=0.9), bias=0.1)(0:100/100)/255
> cols <- mapply(rgb, cols[, 1], cols[, 2], cols[, 3])
> image(m[O, O[dim(m)[2]:1]], yaxt='n', xaxt='n', col=cols,
        xlab="Probe #1 Target Site Position", ylab="Probe #2 Target Site Position")
> axis(1, seq(0, 1, 0.01), probes$start[o[1:s]][O[seq(1, s, length.out=101)]])
> axis(2, seq(1, 0, -0.01), probes$start[o[1:s]][O[seq(1, s, length.out=101)]])
```

There are many combinations of two probes that have fairly similar scores, so now the challenge becomes how to choose the best two probes. One criteria is that the first and second probe's target sites must be separated by at least 50 nucleotides to mitigate helper-oligonucleotide effects. Other criteria include the number of probe permutations required, percent coverage of the group, and individual probe's scores. Next we will take these considerations into account when choosing the top dual-probe sets.

```
> # choose the best probe sets
> d <- dimnames(m)
> p <- outer(probes$permutations[o[1:s]],
        probes$permutations[o[1:s]],
        FUN="+")
> c <- -1*outer(rowSums(as.matrix(probes$coverage[o[1:s],]), na.rm=TRUE),
        ifelse(rep(s, s)==1,
                sum(probes$coverage[o[1:s],], na.rm=TRUE),
                rowSums(as.matrix(probes$coverage[o[1:s],]), na.rm=TRUE)),
```

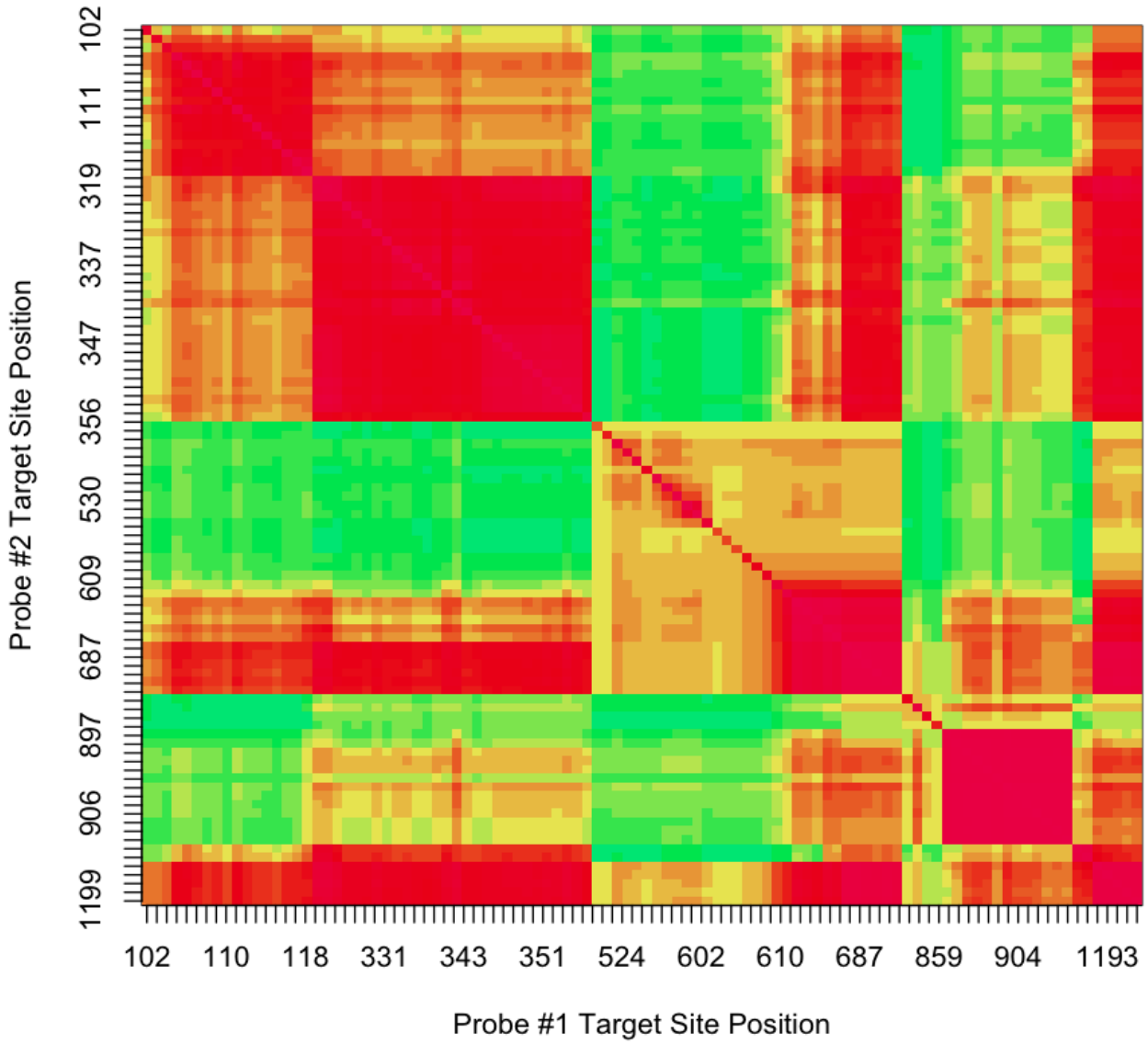

Figure 2: The Matrix of Dual Probe Scores

```
        FUN="*")
> ss <- -1*outer(probes$score[o[1:s]],
        probes$score[o[1:s]],
        FUN="+")
> # order probes by dual-probe score, permutations, coverage, and individual scores
> o <- order(m + n/5, p, c, ss) # score = 0.2*n + 1.2^dFAm
> j <- 0
> first <- second <- integer()
> scores <- numeric()
> mismatches <- character()
> for (k in 1:numProbeSets) {
        if ((j + 1) > length(o))
                break
        for (j in (j + 1):length(o)) {
                w_o <- c((o[j] - 1) %% dim(m)[1] + 1,
                        (o[j] - 1)%/% dim(m)[1] + 1)
                f <- as.numeric(d[[1]][w_o[1]])
                r <- as.numeric(d[[2]][w_o[2]])

                start_F <- probes$start[f]
                start_R <- probes$start[r]
                if (abs(start_F - start_R) > (50 + nchar(probes$probe[f][1])))
                        break # > 50 nt separation
        }

        if (abs(start_F - start_R) > (50 + nchar(probes$probe[f][1]))) {
                first <- c(first, f)
                second <- c(second, r)
                scores <- c(scores, -m[o[j]]/100)

                w_F <- which(index==w_o[1])
                w_R <- which(index==w_o[2])
                overlap_MMs <- match(MMs[w_F], MMs[w_R])
                w_S <- which(!is.na(overlap_MMs))
                if (length(w_S)==0)
                        next
                EFFs <- tapply(c(effs[w_F[w_S]], effs[w_R[overlap_MMs[w_S]]]),
                                rep(1:length(w_S), 2),
                                min)
                # record set mismatches
                w1 <- which(EFFs >= -20)
                if (length(w1) > 0)
                        mismatches <- c(mismatches, paste(MMs[w_F][w_S][w1],
                                " (", formatC(EFFs[w1], digits=1, width=1, format="f"),
                                "%)", sep="", collapse=", "))
        }
 }
> pSets <- cbind(probes[first,], probes[second,])
> ns <- names(pSets)
> ns[1:10] <- paste(ns[1:10], "one", sep="_")
> ns[11:20] <- paste(ns[11:20], "two", sep="_")
```

```
> names(pSets) <- ns
> pSets$score_set <- scores
> pSets$mismatches_set <- mismatches
> pSets[1, -which(names(pSets) %in% c("mismatches_one", "mismatches_two"))]


    identifier_one start_one start_aligned_one permutations_one
653   Sphingopyxis       686               844                2
       score_one               probe_one.1               probe_one.2
653 -50.1945.... CCCGGACAGCTAGTTATCATCGTTTA CCCGGACAGATAGTTATCATCGTTTA
    probe_one.3 probe_one.4 efficiency_one.1 efficiency_one.2
653        <NA>        <NA>        0.8086710        0.6370537
    efficiency_one.3 efficiency_one.4 FAm_one.1 FAm_one.2 FAm_one.3
653               NA               NA  45.12303  38.95117        NA
    FAm_one.4 coverage_one.1 coverage_one.2 coverage_one.3
653        NA            0.8            0.2             NA
    coverage_one.4 identifier_two start_two start_aligned_two
653             NA   Sphingopyxis      1195               1376
    permutations_two    score_two         probe_two.1 probe_two.2
653                1 -44.1990.... CCGCCTTCATGCTCTCGAG        <NA>
    probe_two.3 probe_two.4 efficiency_two.1 efficiency_two.2
653        <NA>        <NA>        0.6540405               NA
    efficiency_two.3 efficiency_two.4 FAm_two.1 FAm_two.2 FAm_two.3
653               NA               NA  40.71256        NA        NA
    FAm_two.4 coverage_two.1 coverage_two.2 coverage_two.3
653        NA              1             NA             NA
    coverage_two.4 score_set
653             NA 0.1546974

653 Porphyrobacter (-4.4%), Novosphingobium (0.0%), Sphingomonas (0.0%),
unclassified_Sphingomonadaceae (0.0%), unclassified_Sphingomonadales (-19.3%),
ref_ref_Altererythrobacter (5.7%), ref_ref_Croceicoccus (5.7%),
ref_ref_Erythrobacter (5.7%), ref_ref_Porphyrobacter (1.3%), ref_ref_Blastomonas
(5.7%), ref_ref_Novosphingobium (5.7%), ref_ref_Sandaracinobacter (-9.3%),
ref_ref_Sandarakinorhabdus (-3.9%), ref_ref_Sphingobium (-0.4%),
ref_ref_Sphingomonas (5.7%), ref_ref_Sphingopyxis (5.7%), ref_ref_Sphingosinicella
(-5.2%), ref_ref_Stakelama (5.7%), ref_ref_Zymomonas (5.7%)
```

There are now multiple candidate dual probe sets for consideration, all of which target variable regions in the alignment. Notice that the top dual probe set does not include the best scoring single probe. Using two probes in conjunction was sufficient to eliminate many non-targets, which are listed in the column *mismatches_set*. However, even using two probes several non-targets still remain, and careful consideration should be given to which probe set has the least liklihood of causing a false positive in the sample. For example, several of the potential non-targets are rarely found in drinking water and pose little threat of causing a false positive identification for this sample.

5.2 Visualizing the Target Sites

Often it is useful to visualize the target sites where the probes will hybridize on both the target and non-targets side-by-side. We can accomplish this by highlighting the region of both probes in the sequence set with `BrowseSeqs`. After querying the database for the sequences of interest, we can selectively color the

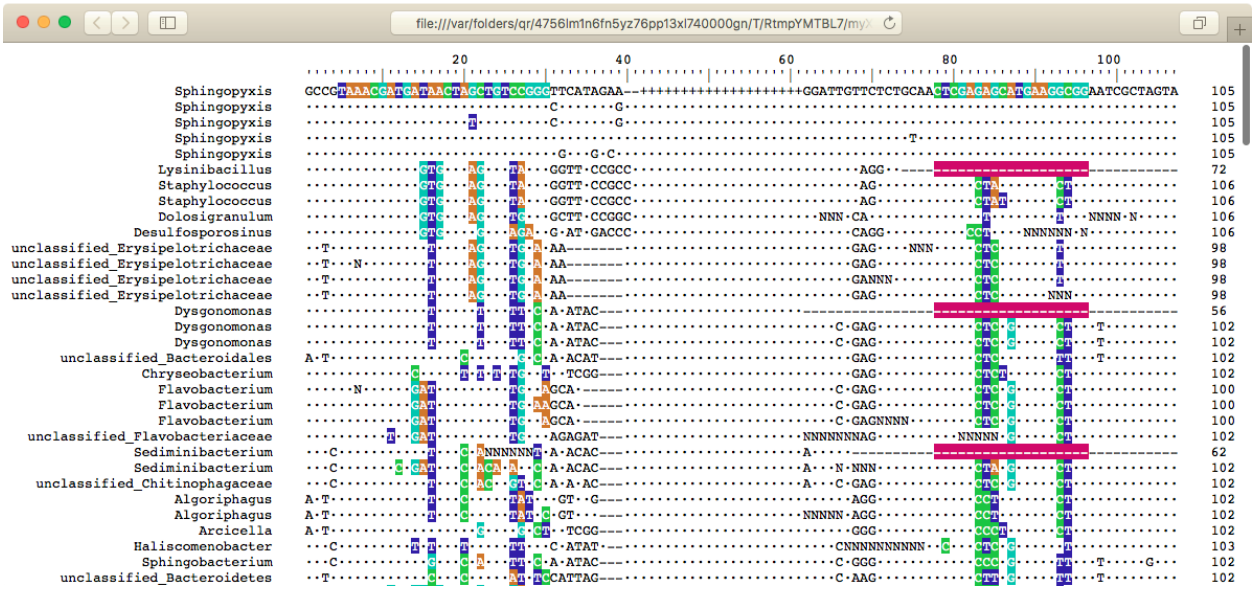

Figure 3: 16S sequences with target site for each probe colored

target site regions given by `start_aligned_one` and `start_aligned_two` outputs of `DesignProbes`. Next we name and order the sequences in the set such that the target (*Sphingopyxis*) appears at the top.

```
> dna <- SearchDB(dbConn, nameBy="identifier", verbose=FALSE)
> dbDisconnect(dbConn)
> # move the target group to the top of the sequence set
> w <- which(names(dna)=="Sphingopyxis")
> dna <- c(dna[w], dna[-w])
> BrowseSeqs(dna, colorPatterns=c(pSets$start_aligned_one[1],
          pSets$start_aligned_one[1] + nchar(pSets$probe_one[1]) - 1,
          pSets$start_aligned_two[1],
          pSets$start_aligned_two[1] + nchar(pSets$probe_two[1]) - 1),
          highlight=1)
```

Examining the output, it is clear why these target sites were chosen for the dual probes: at least one of the dual probes has a number of mismatches to each of the non-target groups.

5.3 Finishing Up

Finally, we can order the first and second probe and try them out in an experiment! The probes should be labeled with different fluorphores (e.g., fluorescein and Cy3). In cases where all non-targets cannot be eliminated, it may be possible to further increase specificity by using a competitor oligonucleotide complementary to the non-target permutations given in the *mismatches* columns. Hybridization conditions could be further optimized based on the results of a formamide gradient experiment. If the two probes have markedly different optimal formamide concentrations then a dual-hybridization approach can be employed by washing with the more stringent condition followed by the second.

6 Session Information

All of the output in this vignette was produced under the following conditions:

- R version 3.6.0 (2019-04-26), `x86_64-pc-linux-gnu`
- Running under: `Ubuntu 18.04.2 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.9-bioc/R/lib/libRblas.so`
- LAPACK: `/home/biocbuild/bbs-3.9-bioc/R/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: BiocGenerics 0.30.0, Biostrings 2.52.0, DECIPHER 2.12.0, IRanges 2.18.0, RSQLite 2.1.1, S4Vectors 0.22.0, XVector 0.24.0
- Loaded via a namespace (and not attached): DBI 1.0.0, KernSmooth 2.23-15, Rcpp 1.0.1, bit 1.1-14, bit64 0.9-7, blob 1.1.1, compiler 3.6.0, digest 0.6.18, memoise 1.1.0, pkgconfig 2.0.2, tools 3.6.0, zlibbioc 1.30.0