Alternative CDF environments

Laurent Gautier

May 3, 2016

Contents

1 Introduction

On short oligonuleotide arrays, several probes are designed to match a target transcript, and probes matching the same target transcript can be grouped in a probe set. Between the time the probes for a given short oligonucleotide chip were designed, and the time an analysis is made, the knowledge of expected transcripts for a given organism might have changed. Unless one includes the latest development in transcripts into an analysis, the analysis could suffer from what we like to call a *Dorian Gray*[1] effect. The chip itself does not change, which means that the probes and their respective sequences remain the same, while the knowledge of the transcripts, and eventually their sequence, might evolve, and in time the immobility of the probe and probe sets give an uglier picture of the biological phenomena to study. Being able to easily modify or replace the grouping of probes in probe sets gives the opportunity to minimize this effect.

The package is directly usable with *Affymetrix GeneChip* short oligonucleotide arrays, and can be adapted or extended to other platforms.

The bibliographic reference associated with the package is given by the command:

```
citation(package="altcdfenvs")
```

> Alternative mapping of probes to genes for Affymetrix chips Laurent Gautier, Morten Mooller, Lennart Friis-Hansen, Steen Knudsen BMC Bioinformatics 2004, 5:111

If you use it, consider citing it, and if you cite it consider citing as well other packages it depends on.

To start we will first load the package:

```
> library(altcdfenvs)
```

[1] From the novel 'The Picture of Dorian Gray' by Oscar Wilde.

2 The class `CdfEnvAffy`

Each instance of this class contains a way to group probes in probe sets. Different instances, describing different ways to group probes in probe sets, can co-exist for a given chip type.

When experimenting, it is highly recommended to use the functions `validCdfEnvAffy` and `validAffyBatch` to make sure that a given instance is a valid one.

3 Reading sequence information in FASTA connections

The package contains simple functions to read **R** connections in the FASTA format. Typically, collections of sequences are stored in FASTA files, which can be significantly large, one can wish to read and process sequences one after the other. This can be done by opening the file in 'r' mode:

```
> fasta.filename <- system.file("exampleData", "sample.fasta",
+                              package="altcdfenvs")
> con <- file(fasta.filename, open="r")
```

Reading the sequences one after another, and printing information about them in turn goes like:

```
> fasta.seq <- read.FASTA.entry(con)
> while(! is.null(fasta.seq$header)) {
+   print(fasta.seq)
+   fasta.seq <- read.FASTA.entry(con)
+ }

FASTA sequence:
   >gnl|UG|Hs#S1730546 membrane-spanning 4-domains, subfamily A ...
   AACCCATTTCAACTGCCTATTCAGAGCATGCAGTAAGAGGAAATCCACCAAGTCTCAATA ...
FASTA sequence:
   >gi|28626515|ref|NM_007257.3| Homo sapiens paraneoplastic an ...
   GGTCATTTGTCCAGAAAACTTTGTGACTGTCTTTGAGTGACCTAGTCTGGGACCCATTCA ...
FASTA sequence:
   >gi|31377729|ref|NM_020143.2| Homo sapiens putatative 28 kDa ...
   TGGCTTCTGCGTGGTGCAGCTGCGCACGTGTTTCAGCCGGCAGCGCTTTAAGATTTCCGG ...

> close(con)
```

One can foresee that the matching of a set of reference sequences against all the probes can be parallelized easily: the reference sequences can simply be distributed

across different processors/machines. When working with all the reference sequences in a single large FASTA file, the option `skip` can let one implement a poor man's parallel sequence matching very easily.

4 Creating an alternative mapping from sequences in a FASTA file

4.1 Select the constituting elements

- Chip type: For this tutorial we decide to work with the Affymetrix chip HG-U133A.
- Target sequences: The set of target sequences we use for this tutorial is in the exemplar FASTA file:

```
> ## first, count the number of FASTA entries in our file
> con <- file(fasta.filename, open="r")
> n <- countskip.FASTA.entries(con)
> close(con)
> ## read all the entries
> con <- file(fasta.filename, open="r")
> my.entries <- read.n.FASTA.entries.split(con, n)
> close(con)
```

matching the probes

The package *Biostrings* and the probe data package for HG-U133A are required to perform the matching. The first step is to load them:

```
> library(hgu133aprobe)
>
```

The matching is done simply (one can refer to the documentation for the package *Biostrings* for further details):

```
> targets <- my.entries$sequences
> names(targets) <-  sub(">.+\\|(Hs\\#|NM_)([^[:blank:]\\|]+).+",
+                        "\\1\\2", my.entries$headers)
> m <- matchAffyProbes(hgu133aprobe, targets, "HG-U133A")
>
>
```

4.2 analyzing the matches

When the position of the match between probes and target sequences does not matter, the association can be represented as a bipartite graph.

The method `toHypergraph` will transform an instance of `AffyProbesMatch` into an `Hypergraph`.

```
> hg <- toHypergraph(m)
```

Currently, there are not many functions implemented around hypergraphs, so we convert it to a more common graph.

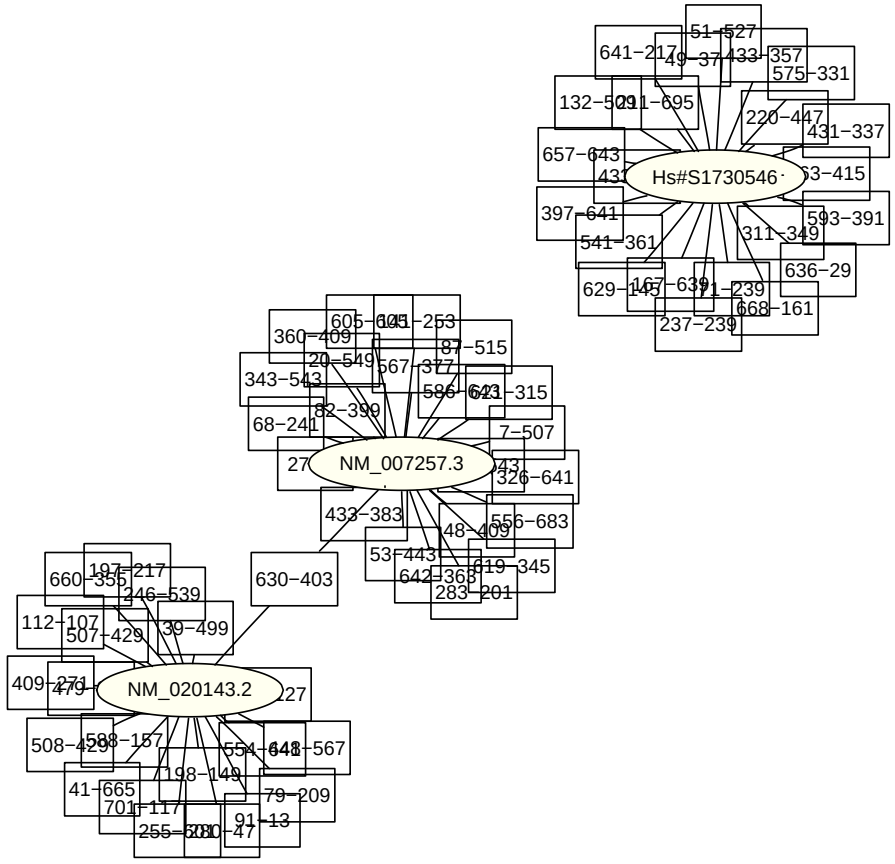
```
> gn <- toGraphNEL(hg)
```

Since this is now a regular graph, all of probes and targets are regular nodes on that graph. Node name-based rules can be applied to identify whether a node is a target sequence or a probe.

```
> targetNodes <- new.env(hash=TRUE, parent=emptyenv())
> for (i in seq(along=targets)) {
+   targetNodes[[names(targets)[i]]] <- i
+ }
```

Since the graph is relatively small, we can plot it, and see that one probe is common to both probe sets:

```
> library(Rgraphviz)
> tShapes <- rep("ellipse", length=length(targets))
> names(tShapes) <- names(targets)
> tColors <- rep("ivory", length=length(targets))
> names(tColors) <- names(targets)
> nAttrs <- list(shape = tShapes, fillcolor = tColors)
> gAttrs <- list(node = list(shape = "rectangle", fixedsize = FALSE))
> plot(gn, "neato",
+      nodeAttrs = nAttrs,
+      attrs = gAttrs)
>
```



Whenever a large number oftarget sequences are involved, counting the degrees will be more efficient than plotting.

The package contains a function to create a `CdfEnv` from the matches:

```
> alt.cdf <-
+   buildCdfEnv.biostrings(m, nrow.chip = 712, ncol.chip = 712)
>
```

Note that the size for chip must be specified. This is currently a problem with cdfenvs as they are created by the package *makecdfenv*. The class `CdfEnv` suggests a way to solve this (hopefully this will be integrated in *makecdfenv* in the near future). When this happens, the section below will be replaced by something more intuitive. But in the meanwhile, here is the current way to use our shiny brand new class `CdfEnv`:

```
## say we have an AffyBatch of HG-U133A chips called 'abatch'

## summary checks to avoid silly mistakes
validAffyBatch(abatch, alt.cdf)
```

```
## it is ok, so we proceed...

## get the environment out of it class
alt.cdfenv <- alt.cdf@envir

abatch@cdfName <- "alt.cdfenv"
```

From now on, the object `abatch` will use our 'alternative mapping' rather than the one provided by the manufacturer of the chip:

```
print(abatch)
```

5 Always up-to-date

Even if alternative mapping is not used upstream of the analysis, it can still be interesting to verify probesets highlighted during data analysis.

The *biomaRt* package makes withdrawing up-to-date sequences very easy, and those sequences can be matched against the probes.

First, we create a *mart*:

```
library(biomaRt)
mart <- useMart("ensembl",dataset="hsapiens_gene_ensembl")
```

(refer to the documentation for the *biomaRt* for further information).

5.1 Casual checking of genes

In this example, we assume that for one reason or an other a researcher would like to know more about the probes matching the SLAMF genes.

```
> geneSymbols <- c("SLAMF1", "SLAMF3", "SLAMF6", "SLAMF7", "SLAMF8", "SLAMF9")
```

The vector `geneSymbols` defined can easily be replaced by your favorite genes; the example below should still work.

We then write a convenience function `getSeq` to extract the sequences. This function appenda a `-<number>` to the HUGO symbol (as there might be several sequences matching).

```
> getSeq <- function(name) {
+   seq <- getSequence(id=name, type="hgnc_symbol",
+                      seqType="cdna", mart = mart)
+
```

```
+   targets <- seq$cdna
+   if (is.null(targets))
+     return(character(0))
+   names(targets) <- paste(seq$hgnc_symbol, 1:nrow(seq), sep="-")
+   return(targets)
+ }
```

The function let us obtain the target sequences very easily:

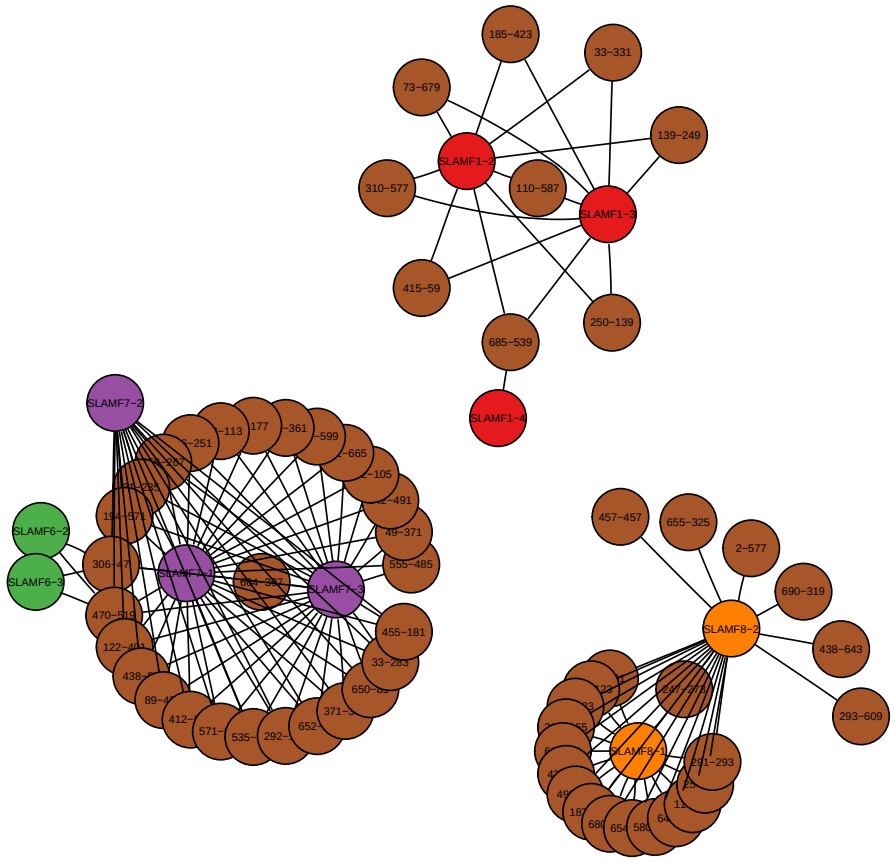
```
targets <- unlist(lapply(geneSymbols,
                         getSeq))
```

The targets are matched as seen previously:

```
> m <- matchAffyProbes(hgu133aprobe, targets, "HG-U133A")
```

A colorful graph can be made in order to visualize how matching probes are distributed:

```
> hg <- toHypergraph(m)
> gn <- toGraphNEL(hg)
> library(RColorBrewer)
> col <- brewer.pal(length(geneSymbols)+1, "Set1")
> tColors <- rep(col[length(col)], length=numNodes(gn))
> names(tColors) <- nodes(gn)
> for (col_i in 1:(length(col)-1)) {
+   node_i <- grep(paste("^", geneSymbols[col_i],
+                        "-", sep=""),
+                        names(tColors))
+   tColors[node_i] <- col[col_i]
+ }
> nAttrs <- list(fillcolor = tColors)
> plot(gn, "twopi", nodeAttrs=nAttrs)
```



- Watch for *SLAMF6* and *SLAMF7*
- The second sequence in SLAMF8 can potentially has specific probes (the rest of the probes are matching both SLAMF8 sequences)

Comparison with the official mapping can be made (not so simply, a future version should address this)

```
> library("hgu133a.db")
> affyTab <- toTable(hgu133aSYMBOL)
> slamf_i <- grep("^SLAMF", affyTab$symbol)
> pset_id <- affyTab$probe_id[slamf_i]
> library("hgu133acdf")
> countProbes <- lapply(pset_id, function(x) nrow(hgu133acdf[[x]]))
> names(countProbes) <- affyTab$symbol[slamf_i]
> countProbes
```

```
$SLAMF1
[1] 11

$SLAMF7
[1] 11

$SLAMF8
[1] 11

$SLAMF8
[1] 11
```

The results do not appear in complete agreement with the matching just performed.