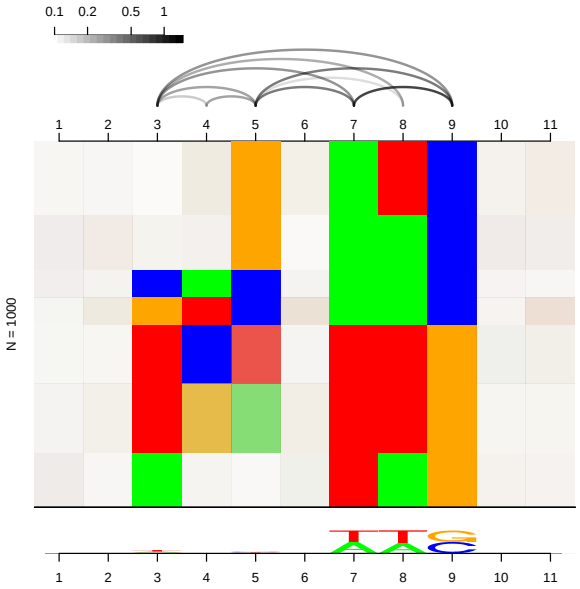
Supplementary information DepLogo: Visualizing sequence dependencies in R

Jan Grau, Martin Nettling and Jens Keilwagen

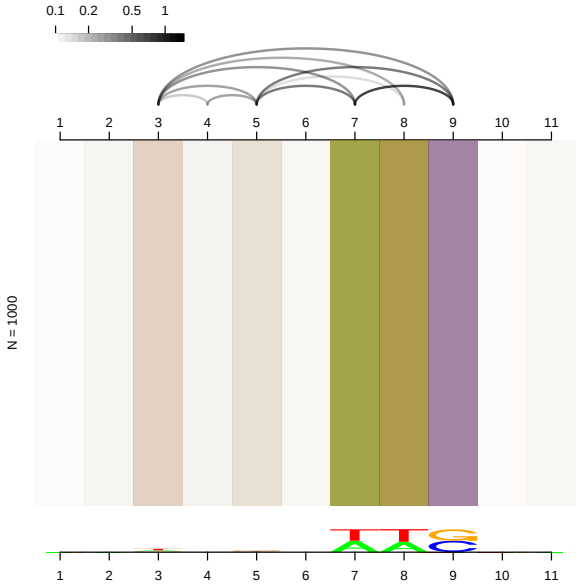
Supplementary Text S1

In the following, we illustrate the process leading to the partitioning that is the basis of dependency logos by means of a step-by-step example. The final dependency logo for the set of example sequences is shown below:



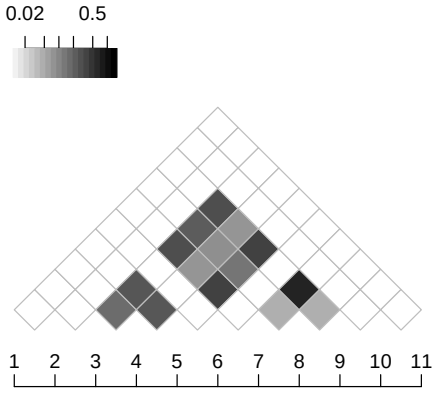
We find several dependencies between sequence positions. Specifically, position 9 is always “C” if position 7 is “A”, while position 9 is “G” if position 7 is “T”. In the subset of sequences that have an “A” at position 7, we also find a dependency between positions 5 and 8, where position 5 is always “G” if position 8 is “T”, whereas for position 8 equal to “A”, position 5 may be “C” or “G”. Similarly, we find a strong dependency between positions 3 and 8 for the subset of sequences with a “T” at position 7. Some of these subsets show further dependencies between positions 3 and 4, and positions 4 and 5, respectively.

We start the partitioning process with the complete set of sequences that is depicted with the same structure as in the dependency logo below:



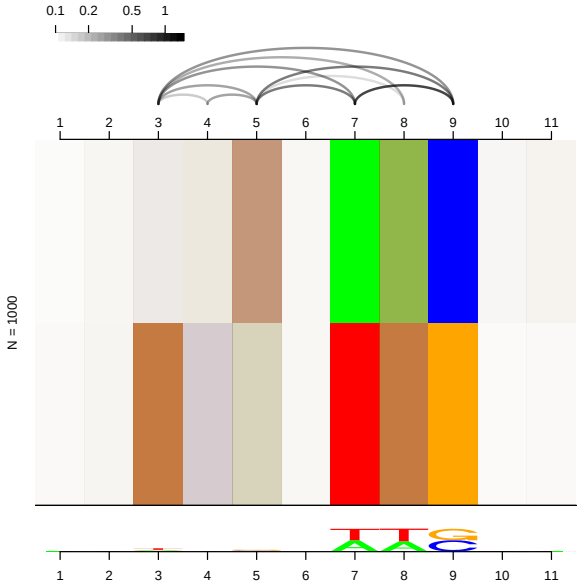
In this representation, we may only spot that positions 7 and 8 are either "A" or "T", and that position 9 is either "G" or "C", but no specific dependency structure is visible.

For choosing the first positions used for partitioning the data, we need to compute all pairwise dependencies between sequence positions. In the `DepLogo` package, dependencies are measured by mutual information. Below, we show a representation of all pairwise mutual information values as obtained by applying the `plotDepmatrix` function of the `DepLogo` package to the complete data set. The mutual information values behind this representation are the same as for the connecting arcs in the dependency logo above.



In the `plotDeplogo` function, the strength of dependencies of one position is computed as the sum of the `numBestForSorting` largest mutual information values to other positions. For reasons of simplicity, we set `numBestForSorting=1` in the following, which means that only the strongest dependency per position is considered. From the matrix representation above, we find the strongest pairwise dependency between positions 7 and 9 as indicated by a black rectangle. These will be the two positions used for the initial partitioning. Specifically, we partition all input sequences by their nucleotides at these very positions. In this specific example, this results in two partitions, because other combinations of nucleotides do not occur in the data. One partition contains all sequences having "A" at position 7 and "C" at position 9, and the other partition contains all sequences having "T" at position 7 and "G" at position 9.

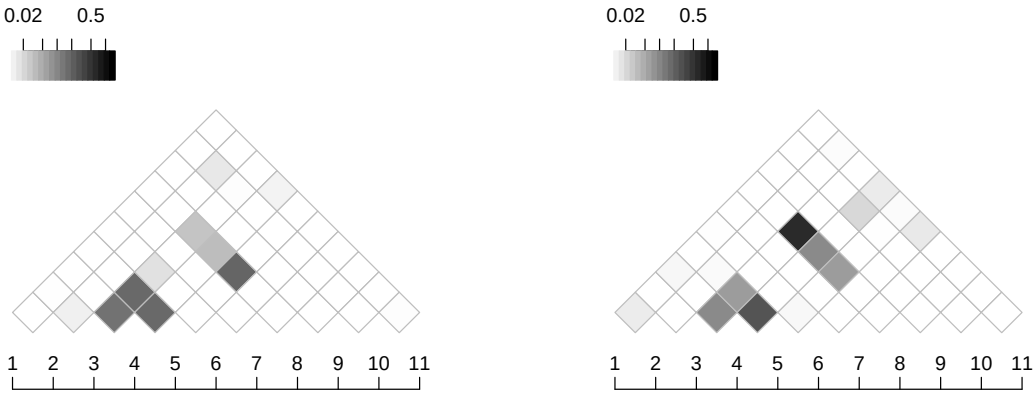
Plotting these partitions in their dependency logo representation, we may already spot this first dependency structure from the visualization:



In addition, we may observe subtle shifts in the colors of some of the other positions (3, 5, 8) that become visible as a result of the partitioning. Brownish colors indicate a shift towards "G"/"T", whereas greenish colors indicate a shift towards "A".

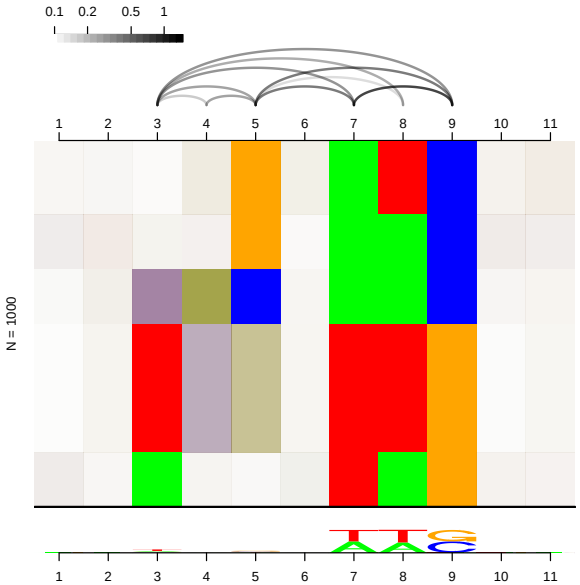
In the `partition` function called by `plotDeplogo` internally, the resulting partitions are partitioned further in a recursive manner. To illustrate this for the example data, we now consider the pairwise dependencies between positions in each of the partitions.

Below, we show the corresponding representations for the first partition (left) and the second partition (right):



Notably, the previous dependency between positions 7 and 9 now disappeared due to the partitioning by the nucleotides at these positions. Instead, we find the strongest dependency between positions 5 and 8 for the first partition, and between positions 3 and 8 for the second partition. The nucleotides at these positions are considered in a recursive partitioning of the respective subsets.

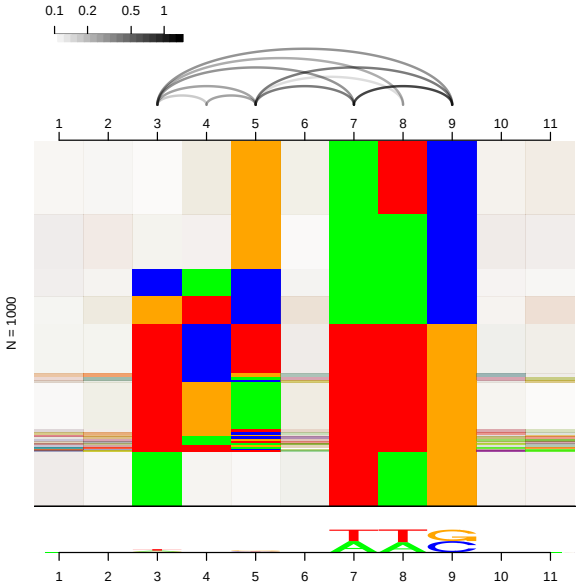
The result of the partitioning is shown in a dependency logo representation below:



We may spot the structure of the dependencies between positions 5 and 8 (first partition) and positions 3 and 8 (second partition) as already explained for the final dependency logo initially.

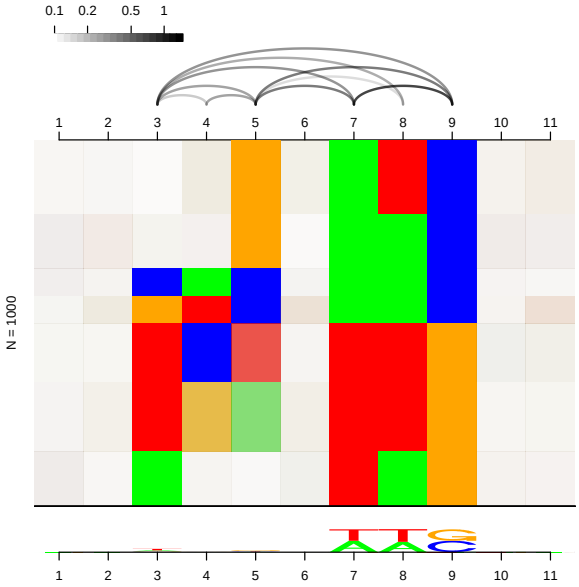
This recursive process continues until a minimum percentage of input sequences is reached. In each partitioning step, it may also happen that one subset of sequences falls below this minimum percentage. In this case, it is merged with the (smallest) partition above the threshold.

For illustration purposes, we set this threshold to the percentage corresponding to a single sequence below:

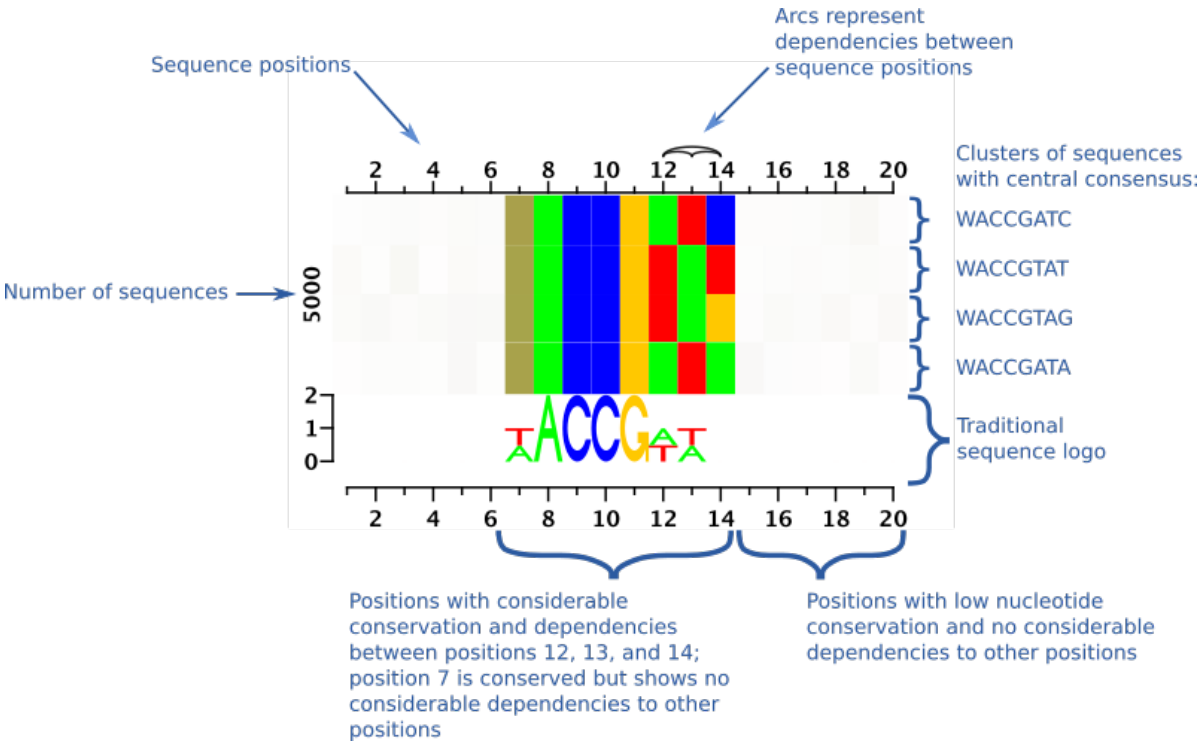


We find some visual clutter in the lower half of the dependency logo representation, which might distract from the more general patterns.

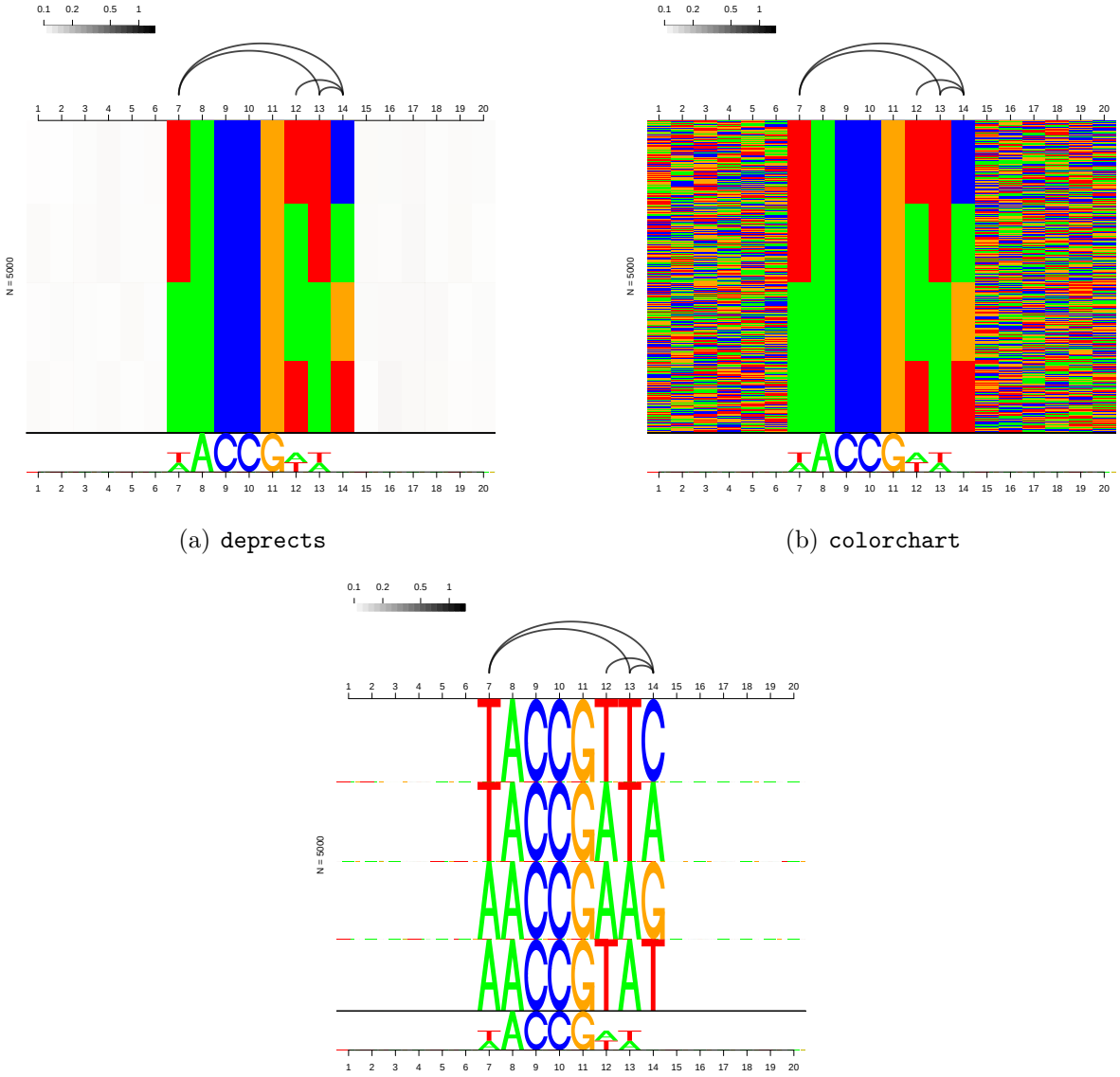
The default value of 0.03 leads to a clearer visual representation as shown below:



Due to the smaller partitions being joined in, the sub-partitions for the sequences showing “T” at positions 3 and 8 are slightly subdued at positions 4 (“C” or “G”) and 5 (“T” or “A”).



Supplementary Figure S1. The basic idea of dependency logos is to show dependencies in (aligned) sequence data by partitioning of input sequences based on the nucleotides occurring at those positions with the strongest dependencies. Each of the resulting partitions is then visualized separately. In case of dependent positions, the symbols occurring at one position are related to those occurring at dependent positions. Hence, a partitioning by symbols at the first position should also lead to a (partial) separation of symbols at those positions depending on the first position. Each partition may be sub-divided into further sub-partitions recursively. At the top of the dependency logo, we find arcs connecting dependent positions. The central part of the plot shows colored boxes representing different partitions of the data. Symbols are represented by colors similar to traditional sequence logos, and the sequence logo at the lower part of the plot helps to recognize the mapping between colors and symbols. On the right of the dependency logo, we annotate partitions with their consensus sequences, where "W" is the IUPAC code for "A or T". Although the dependency logo spans 20 positions, only the central positions 7 to 14 are clearly colored. The reason is that the color assigned to a position of a partition is first mixed, depending on the symbols occurring in that partition at that position (e.g., position 7 is a mix of red (T) and green (A)), and then opacity is assigned based on "information content". This means that, in the same manner as in traditional sequence logos where positions which are close to a uniform distribution are scaled down, color intensity is scaled down at such positions (and partitions) in dependency logos. Based on the dependency logo, we may state the following dependencies for these data i) if position 14 is either "C" or "A" then position 12 is "A" and position 13 is "T", ii) if position 14 is either "T" or "G" then position 12 is "T" and position 13 is "A", and iii) if position 12 is "A" then position 13 is "T", and vice versa.

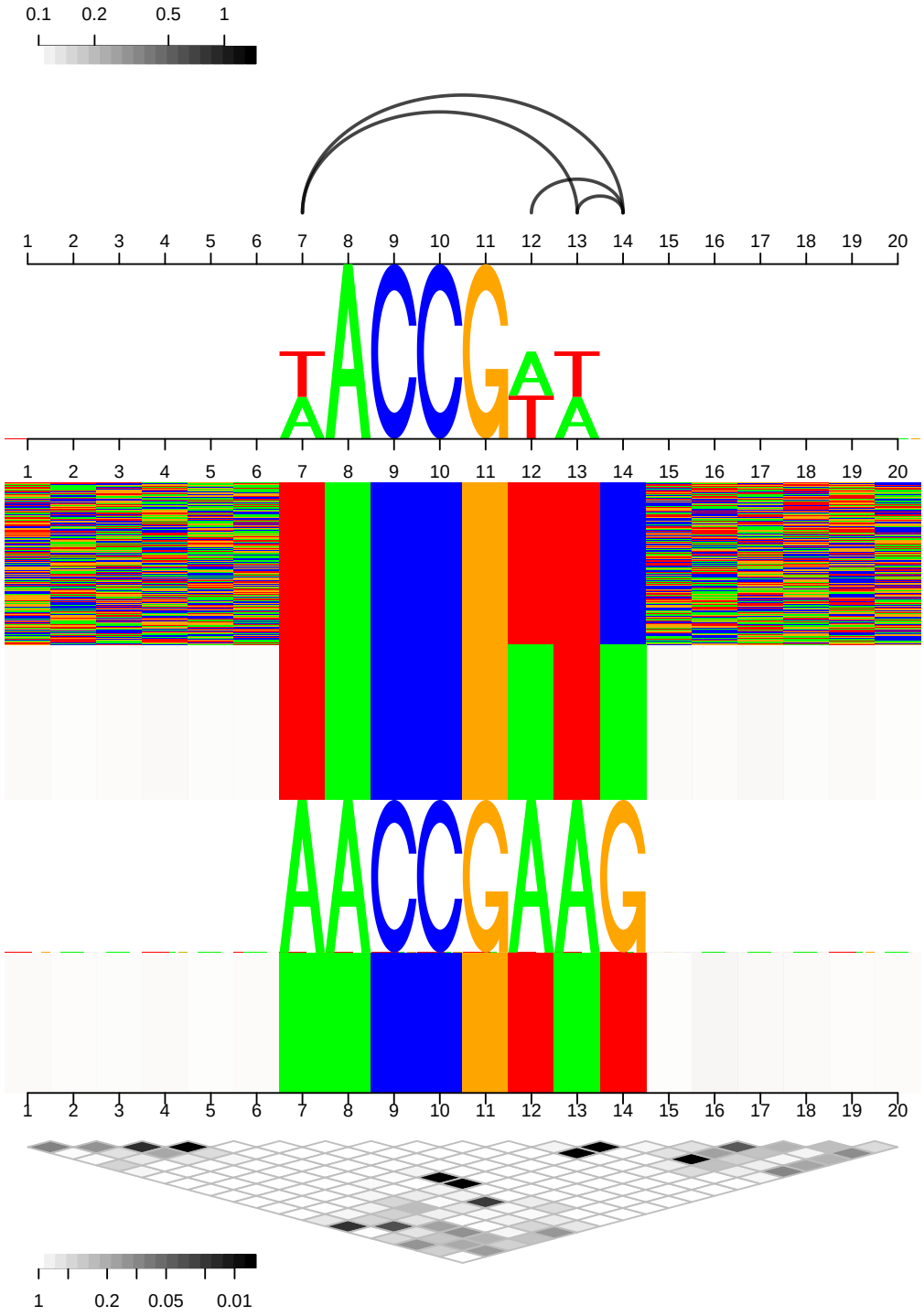


(a) `deprects`

(b) `colorchart`

(c) `logo`

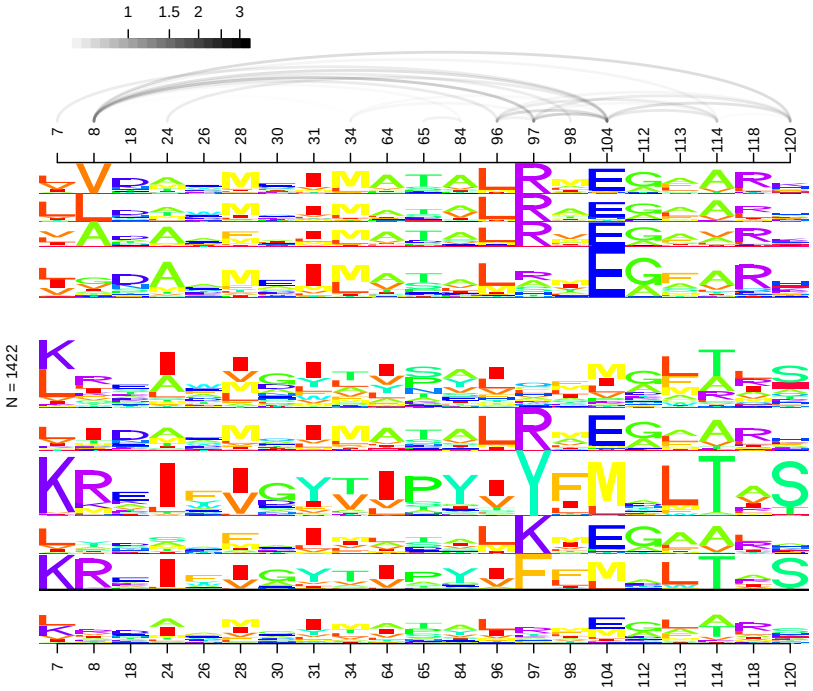
Supplementary Figure S2. Illustration of the different ways of visualizing partitions in `DepLogo` using the `deprects` function that is the default in dependency logos, the `colorchar` function plotting color charts, and the `logo` function plotting sequence logos for each partition.



Supplementary Figure S3. Showcase example of the flexibility of high-level and low-level plotting functions provided by the `DepLogo` package. Here, we first represent the dependencies between positions by connecting arcs using the `plotDeparcs` function, followed by a traditional sequence logo generated by `logo`. The following blocks showcase the different visualizations of the `colorchart`, `deprects`, `logo`, and again `deprects` functions applied to the individual partitions of the input data. Finally, we plot a representation of dependencies as a grid pattern, which is flipped upside-down compared with its standard usage in the `plotDeplogo` function. The code for generating this plot is shown in Supplementary Listing S1.

```
# get the data
seqs <- scan(system.file("extdata", "long_bak.txt",
    package = "DepLogo"), what = "character", comment.char = ">")
data <- DLData(seqs)

par(mar = c(0, 2, 0, 1))

# partion data
parti <- partition(data)

# number of sequence positions
len <- ncol(data) - 1

# layout for the plot
layout( mat = matrix(1:4, ncol = 1), widths = c(1), heights = c(1, 1, 3, 1))

# plot dependencies
plotDeparcs(data, axis.at.bottom = TRUE)

# plot a sequence logo
# as this is a low-level plotting function, we need to
# open a new plot first
plot(NULL, xlim = c(0.5, len + 0.5), ylim = c(0, nrow(data)), axes=FALSE,
    yaxs = 'i', xaxs = "i", xlab = "", ylab = "")
logo(part = data, yoff = nrow(data))
axis(1, at = 1:len)

# total number of sequences
total <- nrow(data)

# plot for the partitions
plot(NULL, xlim = c(0.5, len + 0.5), ylim = c(0, total + 2), axes = FALSE,
    yaxs = 'i', xaxs = "i", xlab = "", ylab = "")

# use different low-level plotting functions
# for the individual partitions
total <- colorchart(part = parti[[1]], yoff = total)
total <- deprects(part = parti[[2]], yoff = total)
total <- logo(part = parti[[3]], yoff = total)
total <- deprects(part = parti[[4]], yoff = total)

par(mar = c(2, 2, 2, 1))
# plot another representation of dependencies at bottom
plotDepmatrix(data, axis.at.bottom = FALSE, show.pvals = TRUE)
```

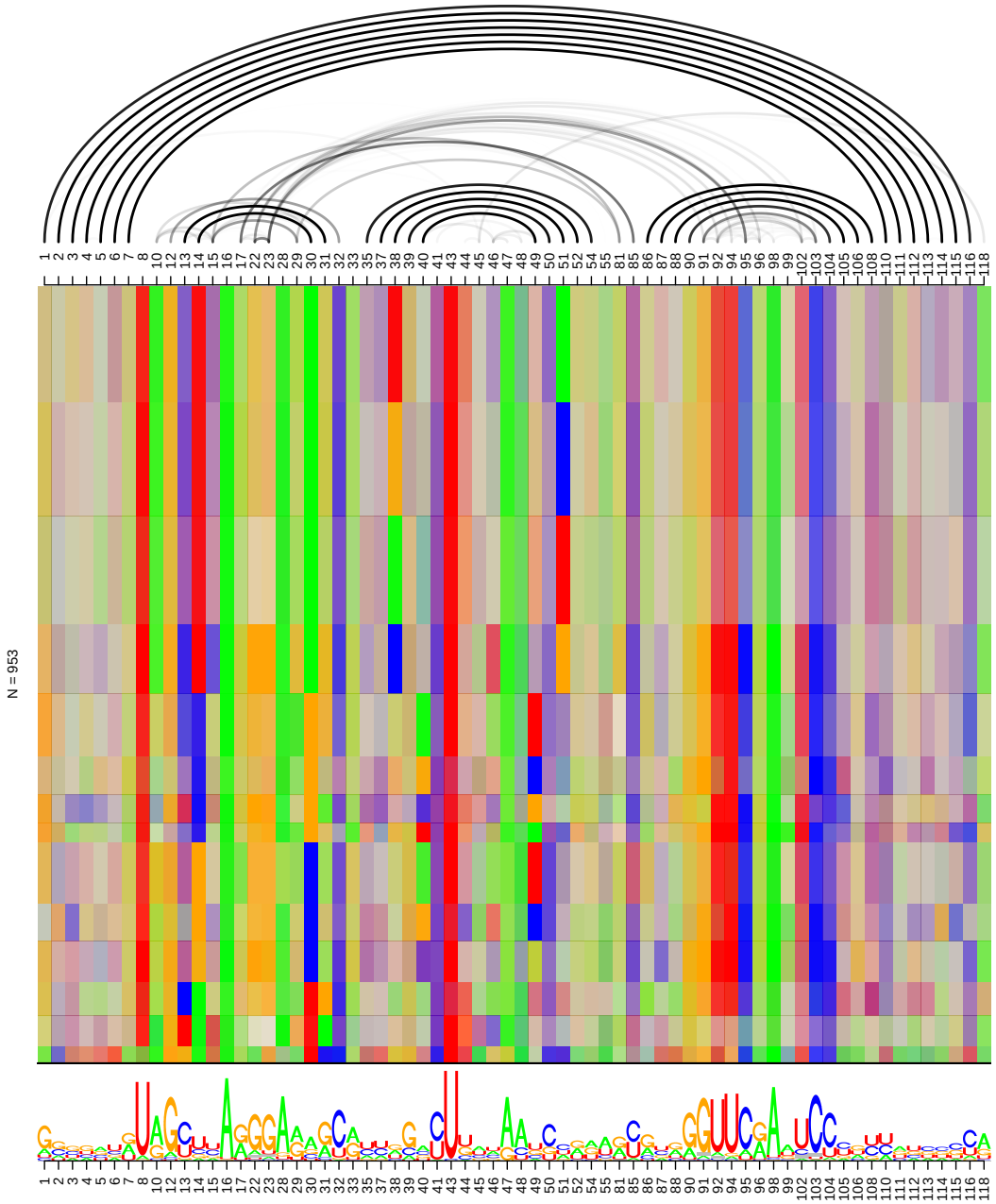
Supplementary Listing S1. Code for generating Supplementary Figure S3.



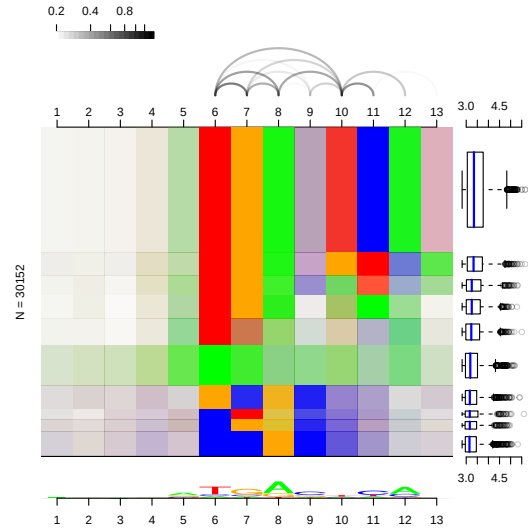
Supplementary Figure S4. Dependency logo of protein sequences of Glycosyl transferases. we find several co-varying positions with complex dependency structures. For instance, position 8 is almost never "R" if position 97 is "R", whereas position 8 may be "R" if position 97 is "Y", "F" or some another amino acid with minor frequency at position 97. The code for generating this plot is shown in Supplementary Listing S2.

```
data <- DLData(sequences = seqs,
  symbols = alphabet.protein.gap$alphabet,
  colors = alphabet.protein.gap$colors)
data.2 <- filterColumns(data = data,
  filter.by.gaps(percent.gap = 0.1))
data.3 <- filterColumns(data = data.2,
  filter.by.dependencies(mi.threshold = 0.2))
data.4 <- replaceColors(data.3,
  suggestColors(data.3))
plotDeplogo(data.4, threshold = 0.55,
  block.fun = logo, minPercent = 0.05)
```

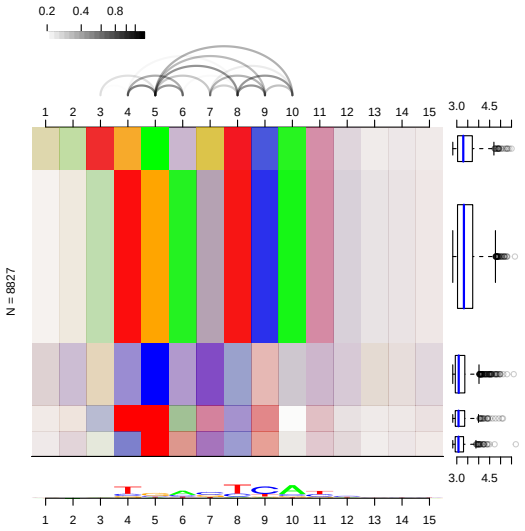
Supplementary Listing S2. Code for generating the dependency logo shown in Supplementary Figure S4. Starting from the protein sequences stored in the variable `seqs`, we create a `DLData` object and filter columns by the fraction of gaps and for the most dependent position. We further replace intially chosen colors by those suggested by `DepLogo` and join partitions containing less then 5% of the sequences. As colors for 20 different amino acids might still be hard to distinguish, we plot the partitions as sequence logos in this case.



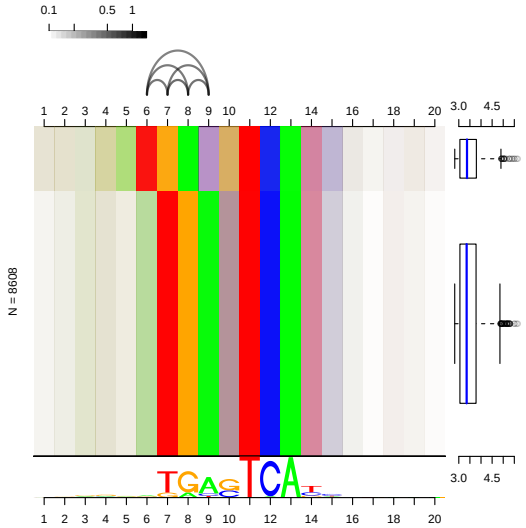
Supplementary Figure S5. Dependency logo of tRNA sequences from RFAM (RF00005). We clearly see the dependencies of the base-pairs forming the clover leaf structure, while co-variation of nucleotides is visible at core positions. For instance, position 30 is "A" if position 14 is "U", whereas position 30 is "G" if position 14 is "C", which corresponds to the complementary base pairing found in RNA sequences. Deeper insights into co-variation structures could be gained by sub-plots of user-selected columns around the stem loop structures.



(a) Bayesian Markov model



(b) Parsimonious Markov model



(c) Slim model

Supplementary Figure S6. Dependency logos visualizing the predictions of three tools using motifs models that are capable of capturing dependencies. Specifically, we consider predictions of a Bayesian Markov model (a), a parsimonious Markov model (b), and a Slim model (c) applied to ChIP-seq data of the human transcription factor c-Jun. We find that all three tools detect the variable spacer of the c-Jun motif, which may be perceived as a shifted pattern between different partitions. For the Bayesian Markov model, we find this shifted patterns between the top two to three partitions at positions 10-12 and 11-13, respectively. For the parsimonious Markov model, we find a similar pattern in the first two partitions, but in a reverse complementary orientation at positions 3-5 and 4-6, respectively. For the Slim model, the shift occurs between the only two partitions at positions 6-8 and 7-9, respectively. In contrast to the Slim model, the Bayesian Markov model and the parsimonious Markov model included further motif sub-types into their predictions, which are visible at the bottom of both dependency logos. From the boxplots of the corresponding ChIP-seq statistics, we recognize, however, that these other motif types receive slightly lower values than the genuine c-Jun motif variants. Notably, the traditional sequence logos for the Bayesian Markov model and the parsimonious Markov model are rather subdued due to the heterogeneities in the predictions of these models, while motif patterns are clearly visible in the dependency logo.