The Art of Multiple Sequence Alignment in R

Erik S. Wright

September 30, 2026

Contents

1 Introduction **1**

2 Alignment Speed **2**

3 Alignment Accuracy **4**

4 Recommendations for optimal performance **7**

5 Single Gene Alignment **8**
5.1 Example: Protein coding sequences 8
5.2 Example: Non-coding RNA sequences 9
5.3 Example: Aligning two aligned sequence sets 9

6 Advanced Options & Features **10**
6.1 Example: Building a Guide Tree 10
6.2 Example: Post-processing an existing multiple alignment 12

7 Aligning Homologous Regions of Multiple Genomes **12**

8 Session Information **15**

1 Introduction

This document is intended to illustrate the art of multiple sequence alignment in R using DECIPHER. Even though its beauty is often concealed, multiple sequence alignment is a form of art in more ways than one. Take a look at Figure 1 for an illustration of what is happening behind the scenes during multiple sequence alignment. The practice of sequence alignment is one that requires a degree of skill, and it is that art which this vignette intends to convey. It is simply not enough to "plug" sequences into a multiple sequence aligner and blindly trust the result. An appreciation for the art as well a careful consideration of the results are required.

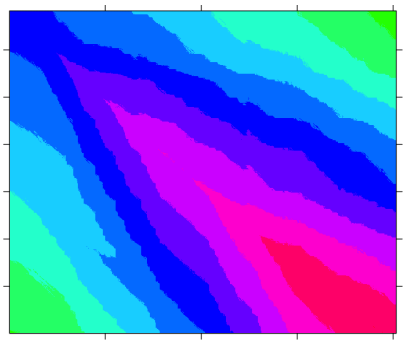

Figure 1: The art of multiple sequence alignment.

What really is multiple sequence alignment, and is there a single correct alignment? Generally speaking, alignment seeks to perform the act of taking multiple divergent biological sequences of the same "type" and fitting them to a form that reflects some shared quality. That quality may be how they look structurally, how they evolved from a common ancestor, or optimization of a mathematical construct. As with most multiple sequence aligners, DECIPHER is "trained" to maximize scoring

metrics in order to accomplish a combination of both structural alignment and evolutionary alignment. The idea is to give the alignment a biological basis even though the molecules that the sequences represent will never meet each other and align under any natural circumstance.

The workhorse for sequence alignment in DECIPHER is `AlignProfiles`, which takes in two aligned sets of DNA, RNA, or amino acid (AA) sequences and returns a merged alignment. For more than two sequences, the function `AlignSeqs` can be used to perform multiple sequence alignment in a progressive/iterative manner on sequences of the same kind. In this case, multiple alignment works by aligning two sequences, merging with another sequence, merging with another set of sequences, and so-forth until all the sequences are aligned. This process is iterated to further refine the alignment. There are other functions that extend use of `AlignSeqs` for different purposes:

1. The first is `AlignTranslation`, which will align DNA/RNA sequences based on their amino acid translation and then reverse translate them back to DNA/RNA. Aligning protein sequences is more accurate since amino acids are more conserved than their corresponding coding sequence.

2. The second function, `AlignDB`, enables generating alignments from many more sequences than are possible to fit in memory. Its main purpose is to merge sub-alignments where each alignment alone is composed of many thousands of sequences. This is accomplished by storing all of the aligned sequences in a database and only working with "profiles" representing the alignment.

3. The function `AdjustAlignment` takes in an existing alignment and shifts groups of gaps right and left to achieve a better alignment. Its purpose is to eliminate artifacts that accumulate during progressive alignment, and to replace the tedious & subjective process of manually correcting an alignment.

4. Finally, `StaggerAlignment` will create a "staggered" alignment by separating potentially non-homologous positions into separate columns. This function will help minimize false homologies when building a phylogenetic tree, although the resulting alignment is not as aesthetically pleasing.

5. The functions `FindSynteny` and `AlignSynteny` can be used in combination to perform pairwise alignment of homologous regions from multiple genomes or non-collinear sequences. These functions interact with a sequence database containing the genomes, which can each be comprised of multiple sequences (i.e., scaffolds, contigs, or chromosomes).

2 Alignment Speed

The dynamic programming method used by DECIPHER for aligning two profiles requires order `N*M` time and memory space where `N` and `M` are the width of the pattern and subject. Since multiple sequence alignment is an inherently challenging problem for long sequences, heuristics are employed to maximize speed while maintaining reasonable accuracy. In this regard, the two control parameters available to the user are *restrict* and *anchor*. The objective of the *restrict* parameter is to convert the problem from one taking quadratic time to linear time. The goal of the *anchor* parameter is do the equivalent for memory space so that very long sequences can be efficiently aligned.

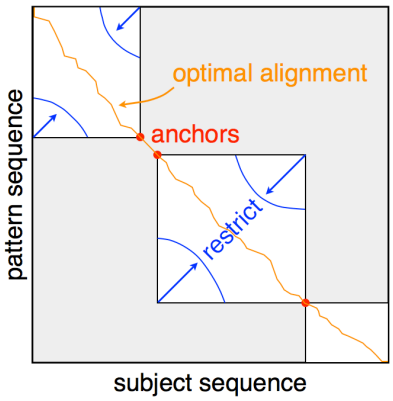

Figure 2: The possible alignment space.

The orange diagonal line in Figure 2 shows the optimal path for aligning two sequence profiles. The blue segments to the left and right of the optimal path give the constraint boundaries, which the user controls with the *restrict* parameter. Areas above and below the upper and lower (respectively) constraint boundaries are neglected from further consideration. A higher (less negative) value of *restrict[1]* will further constrain the possible "alignment space," which represents all possible alignments between two sequences. Since the optimal path is not known till completion of the matrix, it is risky to overly constrain the matrix. This is particularly true in situations where the sequences are not mostly overlapping because the optimal path will likely not be diagonal,

causing the path to cross a constraint boundary. In the non-overlapping case *restrict[1]* could be set below the default to ensure that the entire "alignment space" is available.

Neglecting the "corners" of the alignment space effectively converts a quadratic time problem into a near-linear time problem. We can see this by comparing `AlignProfiles` with and without restricting the matrix at different sequence lengths. To extend our comparison we can include the DECIPHER function `AlignPairs`, which is designed specifically for fast pairwise alignment. In this simulation, two sequences with 90% identity are aligned and the elapsed time is recorded for a variety of sequence lengths. As can be seen in Figure 3 below, *without* restriction `AlignProfiles` takes quadratic time. However, *with* restriction `AlignProfiles` takes linear time, requiring far less than a microsecond per nucleotide.

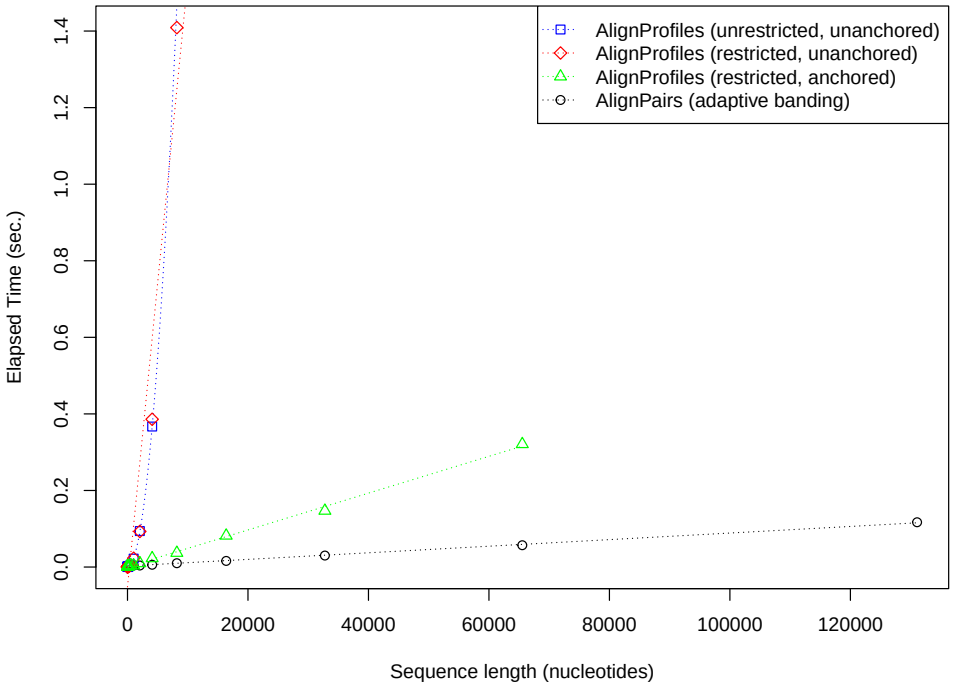

Figure 3: Global Pairwise Sequence Alignment Timings.

The parameter *anchor* controls the fraction of sequences that must share a common region to anchor the alignment space (Fig. 2). `AlignProfiles` will search for shared anchor points between the two sequence sets being aligned, and if the fraction shared is above *anchor* (70% by default) then that position is fixed in the "alignment space." Anchors are 15-mer (for DNA/RNA) or 7-mer (for AA) exact matches between two sequences that must occur in the same order in both sequence profiles. Anchoring generally does not affect accuracy, but can greatly diminish the amount of memory required for alignment. In Fig. 2, the largest white box represents the maximal memory space required with anchoring, while the entire alignment space (grey plus white areas) would be required without anchoring. The longest pair of sequence profiles that can be aligned without anchoring is about 46 thousand nucleotides, as shown by the end of the red dotted line in Figure 3. If regularly spaced anchor points are available then the maximum sequence length is greatly extended. In the vast majority of cases anchoring gives the same result as without anchoring, but with less time and memory space required.

3 Alignment Accuracy

Figure 4 compares the performance of DECIPHER to other sequence alignment software on structural amino acid benchmarks [2]. All benchmarks have flaws, some of which can easily be found by eye in highly similar sequence sets, and therefore benchmark results should treated with care [4]. As can be seen in the figure, the performance of DECIPHER is similar to that of other popular alignment software such as MAFFT [5] and MUSCLE [3] for smaller benchmarks. However, DECIPHER outperforms other programs on large sequence sets (Fig. 5), and its relative advantage continues increase as more sequences are aligned [13]. Importantly, this is because DECIPHER exhibits far less fall-off in accuracy as additional sequences are added.

The accuracy of protein alignment begins to drop-off when sequences in the reference alignment have less than 40% average pairwise identity (Fig. 4). A similar decline in performance is observed with DNA/RNA sequences, but the drop-off occurs much earlier at around 60% sequence identity. Therefore, it is generally preferable to align coding sequences by their translation using `AlignTranslation`. This function first translates the input DNA/RNA sequences, then aligns the translation, and finally (conceptually) reverse translates the amino acid sequences to obtain aligned DNA/RNA sequences. Nevertheless, even protein alignment cannot be considered reliable when the sequences being aligned differ by more than 70%.

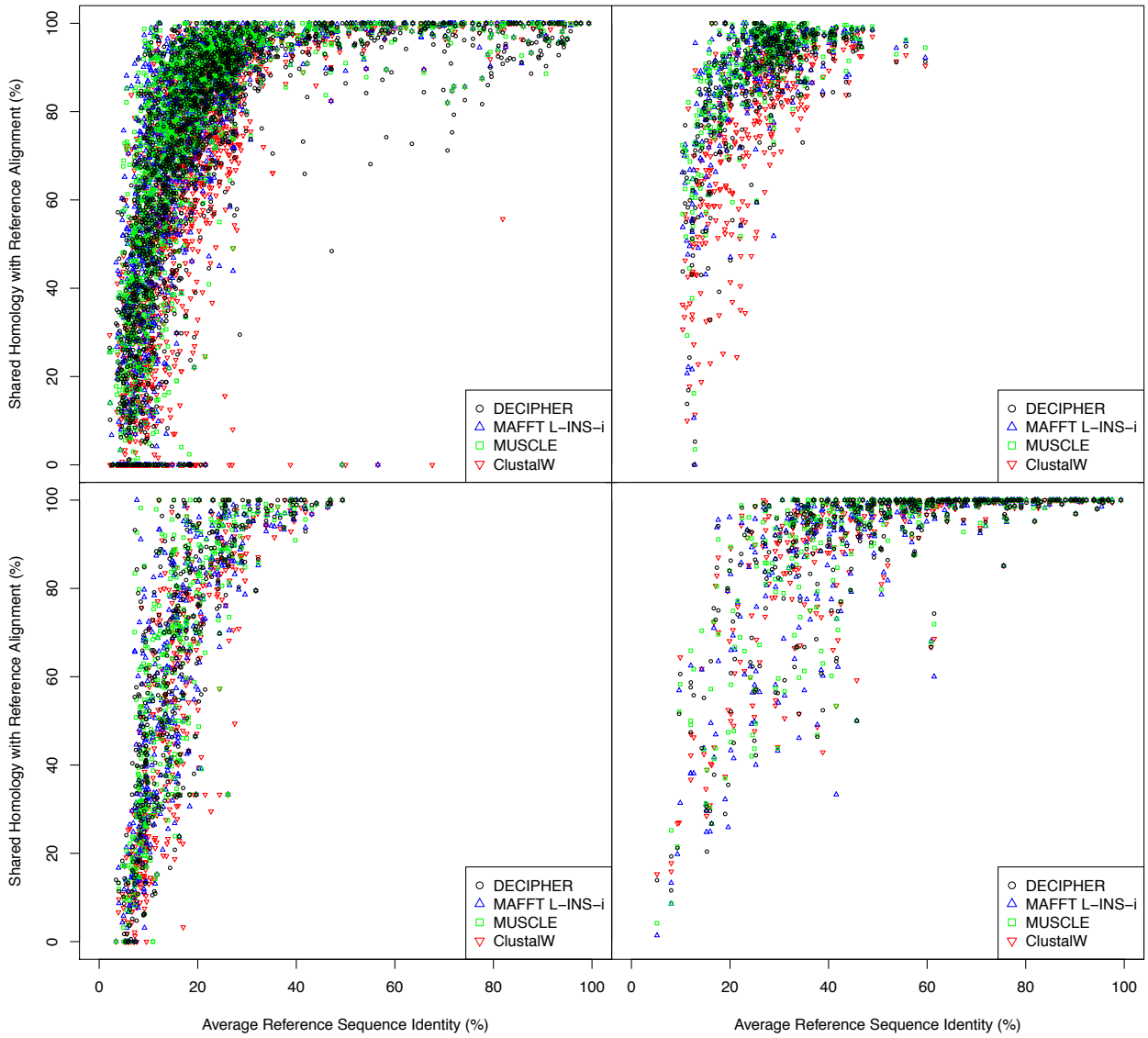

Figure 4: Performance comparison between different programs for multiple alignment [3, 5, 10, 13] using amino acid structural benchmarks. The x-axis shows percent identity between sequences in each reference alignment. The y-axis gives the percentage of correctly aligned residues in the estimated alignment according to the reference alignment (i.e., the Q-score). The upper-left plot is for the PREFAB (version 4) benchmark [3]. The upper-right plot shows the results of the BALiBASE (version 3) benchmark [11]. The lower-left plot is for SABmark (version 1.65) [12]. The lower-right plot gives the results on the OXBench alignments [8]. A comparison of these benchmarks can be found in reference [2].

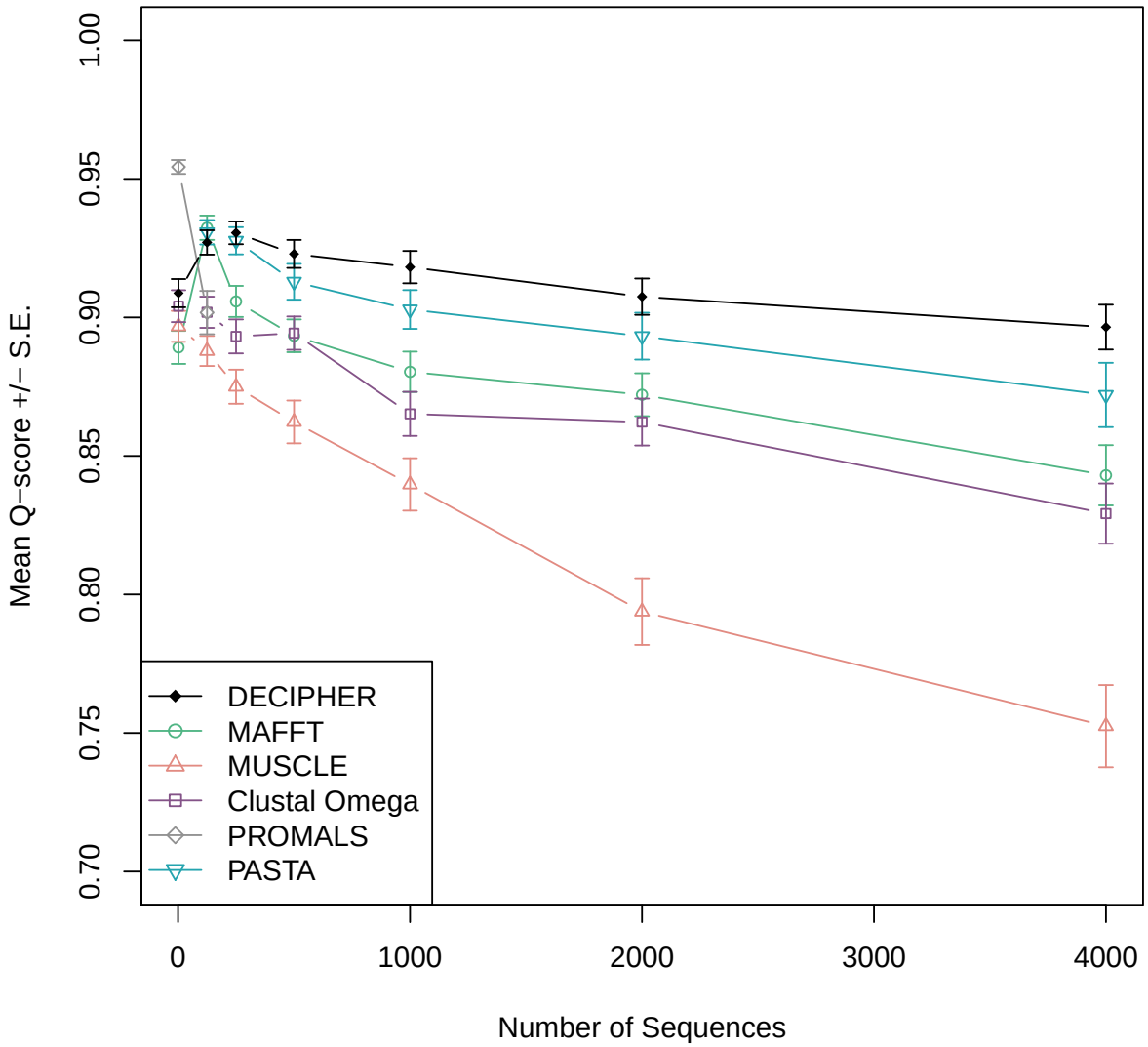

Figure 5: DECIPHER offers improved accuracy over other alignment programs ([3, 5–7, 9, 13]) on large sets of input sequences. Average accuracy on the Homstrad-mod benchmark [13] is shown for an increasing number of input sequences, ranging from 2 to 4,000. All programs display a peak in accuracy at fewer than 500 sequences, but DECIPHER exhibits the least drop-off in accuracy as additional input sequences are added.

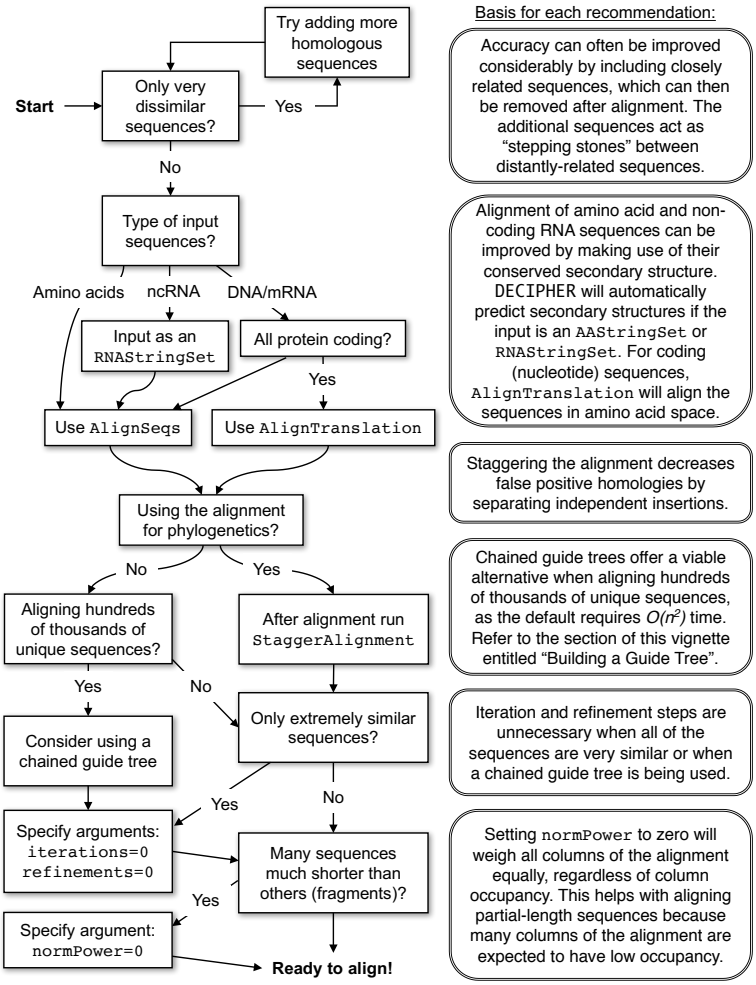

Figure 6: Flow-chart depicting how to choose the best combination of alignment functions and parameters for the most common multiple sequence alignment problems.

4 Recommendations for optimal performance

DECIPHER has a number of alignment functions and associated parameters. The flow-chart in Figure 6 is intended to simplify this process for the most frequently encountered multiple sequence alignment tasks. For more information on any of these suggestions, refer to the examples in the following sections of this vignette.

5 Single Gene Alignment

5.1 Example: Protein coding sequences

For this example we are going to align the *rplB* coding sequence from many different Bacteria. The *rplB* gene encodes one of the primary ribosomal RNA binding proteins: the 50S ribosomal protein L2. We begin by loading the library and importing the sequences from a FASTA file. 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.

```
> library(DECIPHER)
> # specify the path to your sequence file:
> fas <- "<<path to FASTA file>>"
> # OR find the example sequence file used in this tutorial:
> fas <- system.file("extdata", "50S_ribosomal_protein_L2.fas", package="DECIPHER")
> dna <- readDNAStringSet(fas)
> dna # the unaligned sequences
DNAStringSet object of length 317:
      width sequence                                        names
  [1]   819 ATGGCTTTAAAAAATTTTAATC...ATTTATTGTAAAAAAAAGAAAA Rickettsia prowaz...
  [2]   822 ATGGGAATACGTAAACTCAAGC...CATCATTGAGAGAAGGAAAAAG Porphyromonas gin...
  [3]   822 ATGGGAATACGTAAACTCAAGC...CATCATTGAGAGAAGGAAAAAG Porphyromonas gin...
  [4]   822 ATGGGAATACGTAAACTCAAGC...CATCATTGAGAGAAGGAAAAAG Porphyromonas gin...
  [5]   819 ATGGCTATCGTTAAATGTAAGC...CATCGTACGTCGTCGTGGTAAA Pasteurella multo...
  ...   ... ...
[313]   819 ATGGCAATTGTTAAATGTAAAC...TATCGTACGTCGCCGTACTAAA Pectobacterium at...
[314]   822 ATGCCTATTCAAAAATGCAAAC...TATTCGCGATCGTCGCGTCAAG Acinetobacter sp....
[315]   864 ATGGGCATTCGCGTTTACCGAC...GGGTCGCGGTGGTCGTCAGTCT Thermosynechococc...
[316]   831 ATGGCACTGAAGACATTCAATC...AAGCCGCCACAAGCGGAAGAAG Bradyrhizobium ja...
[317]   840 ATGGGCATTCGCAAATATCGAC...CAAGACGGCTTCCGGGCGAGGT Gloeobacter viola...
```

We can align the DNA by either aligning the coding sequences directly, or their translations (amino acid sequences). Both methods result in an aligned set of DNA sequences, unless the argument *type* is `"AAStringSet"` in `AlignTranslation`. A quick inspection reveals that the method of translating before alignment yields a more appealing result. In particular, the reading frame is maintained when aligning the translations. However, if the dna did not code for a protein then the only option would be to use `AlignSeqs` because the translation would be meaningless.

```
> AA <- AlignTranslation(dna, type="AAStringSet") # align the translation
> BrowseSeqs(AA, highlight=1) # view the alignment
> DNA <- AlignSeqs(dna) # align the sequences directly without translation
> DNA <- AlignTranslation(dna) # align the translation then reverse translate
> # write the aligned sequences to a FASTA file
> writeXStringSet(DNA, file="<<path to output file>>")
```

Note that frameshift errors can greatly disrupt the alignment of protein coding sequences. Frameshifts can be corrected by first using `CorrectFrameshifts` on the nucleotide sequences, and then using the corrected sequences as input to `AlignTranslation` with the argument *readingFrame* equal to `1`. It is also feasible to obtain the amino acid alignment or specify a non-standard genetic code, if needed:

```
> # using a mixture of standard and non-standard genetic codes
> gC1 <- getGeneticCode(id_or_name2="1", full.search=FALSE, as.data.frame=FALSE)
> # Mollicutes use an alternative genetic code
> gC2 <- getGeneticCode(id_or_name2="4", full.search=FALSE, as.data.frame=FALSE)
```

```
> w <- grep("Mycoplasma|Ureaplasma", names(dna))
> gC <- vector("list", length(dna))
> gC[-w] <- list(gC1)
> gC[w] <- list(gC2)
> AA <- AlignTranslation(dna, geneticCode=gC, type="AAStringSet")
```

If the input sequences include exact replicates, then alignment can be accelerated by de-replicating the sequences before alignment. The sequences can then be re-replicated after alignment to create a larger alignment of all the original sequences. `AlignSeqs` does not automatically handle redundancy in the input sequences, but doing so is fairly straightforward. In this case there aren't any exact duplicates in the example `dna` sequences. Nonetheless, the process to de-replicate before alignment and re-replicate after alignment would look like:

```
> u_dna <- unique(dna) # the unique input sequences
> index <- match(dna, u_dna) # de-replication index
> U_DNA <- AlignSeqs(u_dna) # align the sequences directly without translation
> DNA <- U_DNA[index]
> names(DNA) <- names(dna) # the re-replicated alignment
```

Also, when aligning nucleotide sequences (or their translations), it may be the case that the sequences are in different orientations. If so, consider reorienting the sequences so that they all have the same directionality and complementarity by using `OrientNucleotides` prior to alignment.

5.2 Example: Non-coding RNA sequences

Much like proteins, non-coding RNAs often have a conserved secondary structure that can be used to improve their alignment. The `PredictDBN` function will predict base pairings from a sequence alignment by calculating the mutual information between pairs of positions. If RNA sequences are given as input, `AlignSeqs` will automatically use the output of `PredictDBN` to iteratively improve the alignment. Providing an *RNAStringSet* also causes single-base and double-base substitution matrices to be used, and is preferable to providing a *DNAStringSet* when the sequences are non-coding RNA. The type of the input sequences can easily be converted to RNA, as shown below.

```
> # database containing 16S ribosomal RNA sequences
> fas <- system.file("extdata", "Bacteria_175seqs.fas", package="DECIPHER")
> dna <- readDNAStringSet(fas)
> rna <- RemoveGaps(RNAStringSet(dna))
> # or if starting with DNA sequences, convert to RNA with:
> # rna <- RNAStringSet(dna)
> # or import RNA sequences directly using:
> # rna <- readRNAStringSet("<<path to FASTA file>>")
>
> alignedRNA <- AlignSeqs(rna) # align with RNA secondary structure
```

5.3 Example: Aligning two aligned sequence sets

It is sometimes useful to align two or more previously-aligned sets of sequences. Here we can use the function `AlignProfiles` to directly align profiles of the two sequence sets:

```
> half <- floor(length(dna)/2)
> dna1 <- dna[1:half] # first half
> dna2 <- dna[(half + 1):length(dna)] # second half
> AA1 <- AlignTranslation(dna1, type="AAStringSet")
> AA2 <- AlignTranslation(dna2, type="AAStringSet")
> AA <- AlignProfiles(AA1, AA2) # align two alignments
```

When the two sequence sets are very large it may be impossible to fit both sets of input sequences and the output alignment into memory at once. The function `AlignDB` can align the sequences in two database tables, or two sets of sequences corresponding to separate *identifier*s in the same table. `AlignDB` takes as input two *tblName*s and/or *identifier*s, and iteratively builds a profile for each of those respective sequence alignments in the database. These profiles are aligned, and the insertions are iteratively applied to each of the input sequences until the completed alignment has been stored in *add2tbl*.

```
> # Align DNA sequences stored in separate tables:
> dbConn <- dbConnect(dbDriver("SQLite"), ":memory:")
> Seqs2DB(AA1, "DNAStringSet", dbConn, "AA1", tblName="AA1")
> Seqs2DB(AA2, "DNAStringSet", dbConn, "AA2", tblName="AA2")
> AlignDB(dbConn, tblName=c("AA1", "AA2"), add2tbl="AA",
         type="AAStringSet")
> AA <- SearchDB(dbConn, tblName="AA", type="AAStringSet")
> BrowseDB(dbConn, tblName="AA")
> dbDisconnect(dbConn)
```

The number of sequences required to fit into memory when aligning two sequence sets with `AlignDB` is controlled by the *batchSize* parameter. In this way `AlignDB` can be used to align large sequence alignments with only minimal memory required.

6 Advanced Options & Features

6.1 Example: Building a Guide Tree

The `AlignSeqs` function uses a guide tree to decide the order in which to align pairs of sequence profiles. The *guideTree* input is a *dendrogram* (tree) object with one leaf per input sequence. By default this guide tree is generated directly from the input sequences using the order of shared k-mers (i.e., when the argument *guideTree* is `NULL`). This default guide tree performs very well but requires $\mathcal{O}(n^2)$ time and memory space to construct. Therefore, it may be useful to rely on a chained guide tree when aligning hundreds of thousands of unique sequences.

It has been shown that reasonably accurate alignments of tens of thousands of sequences can be obtained by using a chain guide tree [1]. With a chained guide tree, sequences are added one-by-one to a growing profile representing all of the aligned sequences. Figure 7 shows the result of using DECIPHER to align increasing numbers of Cytochrome P450 sequences (in accordance with the method in reference [1]), using either a chained guide tree or the default guide tree. A chained guide tree can be easily generated, as shown below.

```
> # form a chained guide tree
> gT <- lapply(order(width(dna), decreasing=TRUE),
         function(x) {
                 attr(x, "height") <- 0
                 attr(x, "label") <- names(dna)[x]
                 attr(x, "members") <- 1L
                 attr(x, "leaf") <- TRUE
                 x
         })
> attr(gT, "height") <- 0.5
> attr(gT, "members") <- length(dna)
> class(gT) <- "dendrogram"
> # use the guide tree as input for alignment
> DNA <- AlignTranslation(dna,
         guideTree=gT,
```

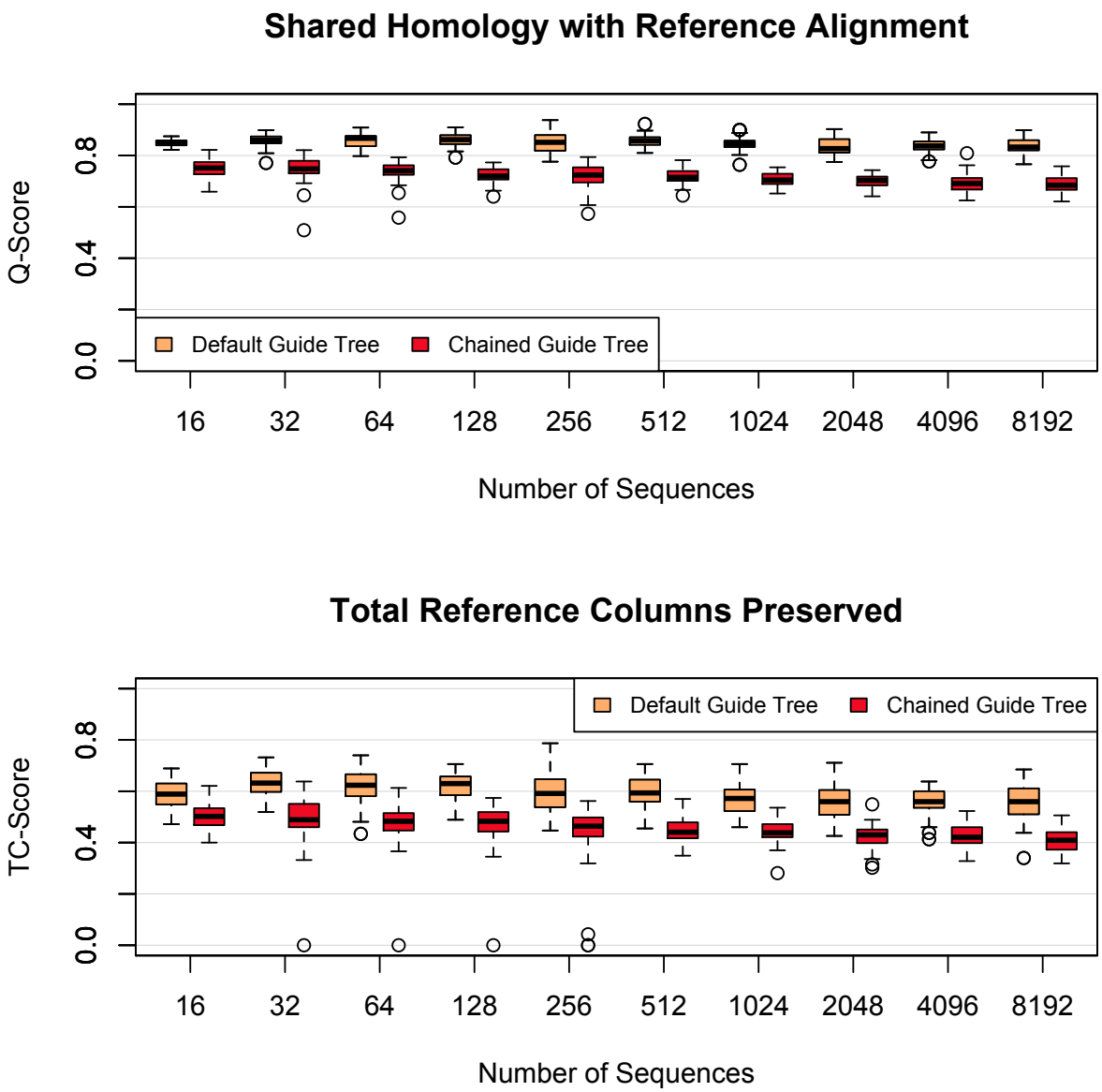

Figure 7: Comparison between the default and chained guide trees when aligning increasing numbers of Cytochrome P450 sequence sets. The top panel shows average pairwise homology shared with the reference alignment (Q-score) and the bottom panel shows the average fraction of alignment columns that are exactly shared with the reference alignment (TC-score).

```
            iterations=0,
            refinements=0)
```

It is also possible to read a Newick formatted tree into R using the function `ReadDendrogram`, and specify this object as the input *guideTree*.

6.2 Example: Post-processing an existing multiple alignment

There are several steps that can be taken after alignment to verify or improve the alignment. The most important step is to look at the result to ensure that it meets expectations. Spurious (unalignable) sequences can then be removed and the alignment process repeated as desired. The simplest way to view sequences with DECIPHER is by using the function `BrowseSeqs`. The *highlight* parameter controls which sequence, if any, is in focus (highlighted). A value of zero highlights the consensus sequence as shown below.

```
> BrowseSeqs(DNA, highlight=0)
```

All DECIPHER multiple sequence alignments are optimized using `AdjustAlignment` (unless the input argument *FUN* is changed), with the goal of removing artifacts of the progressive alignment process. This function will efficiently correct most obvious inaccuracies that could be found by-eye. Therefore, making manual corrections is not recommended unless additional expert knowledge of the sequences is available. The advantage of using `AdjustAlignment` is that it is a repeatable process that is not subjective, unlike most manual adjustments. In order to further refine an existing alignment, `AdjustAlignment` can be called directly.

```
> DNA_adjusted <- AdjustAlignment(DNA)
```

It is common to use alignment as a preliminary step before the creation of a phylogenetic tree. DECIPHER, like the majority of alignment programs, attempts to maximize homologous positions between the sequences being aligned. Such an alignment is particularly useful when investigating which residues are in the same structural position of a protein. However, disparate sequence regions tend to be concentrated into the same "gappy" areas of the alignment. When viewed from a phylogenetic perspective these homologies have highly implausible insertion/deletion scenarios.

To mitigate the problem of false homologies, `StaggerAlignment` will automatically generate a staggered version of an existing alignment. Staggered alignments separate potentially non-homologous regions into separate columns of the alignment. The result is an alignment that is less visually appealing, but likely more accurate from a phylogenetic perspective.

```
> DNA_staggered <- StaggerAlignment(DNA)
```

7 Aligning Homologous Regions of Multiple Genomes

The functions described so far have all required collinear sequences as input. This requirement is frequently broken by genomes, which may include many sequence rearrangements such as inversion, duplication, and reordering. `FindSynteny` will find homologous regions between pairs of genomes, which can then be aligned using `AlignSynteny`. A database of sequences identified by their genome name is used as input to both functions. This enables the alignment of genomes that are composed of many contigs, so long as they all share the same `identifier` in the database. The example below uses a database containing five *Influenza virus A* genomes, which are each composed of eight separate segments.

```
> db <- system.file("extdata", "Influenza.sqlite", package="DECIPHER")
> synteny <- FindSynteny(db, verbose=FALSE)
> synteny # an object of class `Synteny`
> InfluenzaA <- AlignSynteny(synteny, db, verbose=FALSE)
> unlist(InfluenzaA[[1]])
```

The output is a list, with each list component containing a `DNAStringSetList` of pairwise alignments between two genomes. Names of the output correspond to their sequence's `identifier` in the database, and the index of the syntenic block.

It is also possible to display the blocks of synteny between all pairs of genomes. Figure 8 shows the different genome segments (i.e., sequences) separated by thin horizontal and vertical lines. The syntenic blocks are diagonal lines that are composed of many homologous "hits" between the genomes.

```
> pairs(synteny, boxBlocks=TRUE) # scatterplot matrix
```

Figure 8: Dot plots showing the homologous regions among five *Influenza virus A* genomes.

8 Session Information

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

- R version 4.6.1 (2026-06-24), `x86_64-pc-linux-gnu`
- Running under: `Ubuntu 26.04.1 LTS`
- Matrix products: default
- BLAS: `/usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3`
- LAPACK: `/usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.32.so` ; LAPACK version3.12.0
- Base packages: base, datasets, graphics, grDevices, methods, stats, stats4, utils
- Other packages: BiocGenerics 0.59.12, Biostrings 2.81.9, DECIPHER 3.9.4, generics 0.1.4, IRanges 2.47.5, S4Vectors 0.51.10, Seqinfo 1.3.2, XVector 0.53.0
- Loaded via a namespace (and not attached): buildtools 1.0.0, cli 3.6.6, compiler 4.6.1, crayon 1.5.3, DBI 1.3.0, evaluate 1.0.5, knitr 1.52, maketools 1.3.2, rlang 1.3.0, sys 3.4.3, tools 4.6.1, xfun 0.60

References

[1] Boyce, K., Sievers, F., & Higgins, D. G. Simple chained guide trees give high-quality protein multiple sequence alignments. *Proceedings of the National Academy of Sciences of the United States of America.*, 111(29), 10556-10561. doi:10.1073/pnas.1405628111, 2014.

[2] Edgar, R. C. Quality measures for protein alignment benchmarks. *Nucleic Acids Research*, 38(7), 2145-2153. doi:10.1093/nar/gkp1196, 2010.

[3] Edgar, R. C. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5), 1792-97, 2004.

[4] Iantorno, S., Gori, K., Goldman, N., Gil, M., & Dessimoz, C. Who watches the watchmen? An appraisal of benchmarks for multiple sequence alignment. *Methods in Molecular Biology (Clifton, N.J.)*, 1079, 59-73. doi:10.1007/978-1-62703-646-7_4, 2014.

[5] Katoh, K., Misawa, K., Kuma, K.-I., & Miyata, T. MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Research*, 30(14), 3059-3066, 2002.

[6] Mirarab, S., Nguyen, N., Guo, S., Wang, L.-S., Kim, J., & Warnow, T. PASTA: Ultra-Large Multiple Sequence Alignment for Nucleotide and Amino-Acid Sequences. *Journal of Computational Biology*, 22(5), 377-386, 2015.

[7] Pei, J. & Grishin, N. V. PROMALS: towards accurate multiple sequence alignments of distantly related proteins. *Bioinformatics*, 23(7), 802-808, 2007.

[8] Raghava, G. P., Searle, S. M., Audley, P. C., Barber, J. D., & Barton, G. J. OXBench: a benchmark for evaluation of protein multiple sequence alignment accuracy. *BMC Bioinformatics*, 4: 47, 2003.

[9] Sievers, F., Wilm, A., Dineen, D., Gibson, T. J., Karplus, K., Li, W., Lopez, R., McWilliam, H., Remmert, M., Soding, J., Thompson, J., & Higgins, D. Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. *Molecular Systems Biology*, 7, 1-6, 2011.

[10] Thompson, J. D., Higgins, D. G., & Gibson, T. J. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22(22), 4673-4680, 1994.

[11] Thompson, J. D., Koehl, P., Ripp, R., & Poch, O. BAliBASE 3.0: Latest developments of the multiple sequence alignment benchmark. *Proteins*, 61(1), 127-136, 2005.

[12] Van Walle, I., Lasters, I., & Wyns, L. SABmark–a benchmark for sequence alignment that covers the entire known fold space. *Bioinformatics*, 21(7), 1267-1268, 2005.

[13] Wright, E. S. DECIPHER: harnessing local sequence context to improve protein multiple sequence alignment *BMC Bioinformatics*, 16(322), 1-14, 2015.