The Art of Multiple Sequence Alignment in R

Erik S. Wright
University of Wisconsin
Madison, WI

October 13, 2014

Contents

1 Introduction **1**

2 Alignment Speed **2**

3 Alignment Accuracy **3**

4 Example: Single Gene Alignment **4**

5 Example: Building a Guide Tree **4**

6 Example: Aligning Two Aligned Sequence Sets **8**

7 Session Information **8**

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 not simply 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.

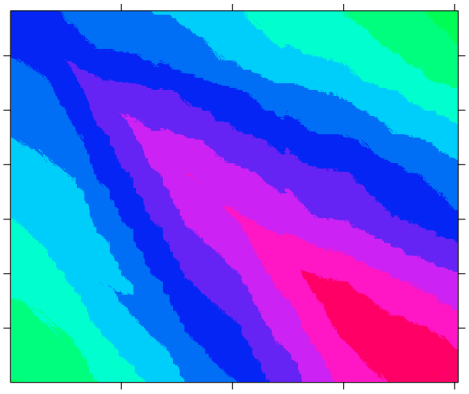

Figure 1: The art of multiple sequence alignment.

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` will 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. This method may improve both alignment accuracy and speed, since amino acid sequences are more conserved, have a well-studied structural basis, and are shorter than their corresponding coding sequences.

2. The second function, `AlignDB`, enables generating alignments from many more sequences than 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 sequences in a database and only working with "profiles" representing the sequences.

2 Alignment Speed

The dynamic programming method 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 large sequences, heuristics are employed to maximize speed while maintaining reasonable accuracy. In this regard, the two most important 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.

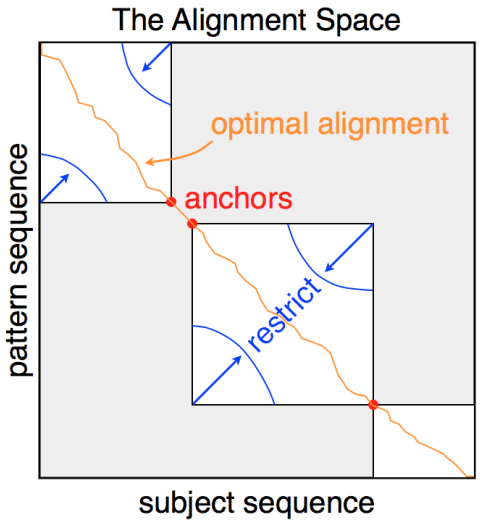

Figure 2: The possible alignment space.

The orange anti-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* 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* should 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 Biostrings function `pairwiseAlignment`. 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 in the same manner as `pairwiseAlignment`. However, *with* restriction `AlignProfiles` takes linear time, requiring only a few microseconds per nucleotide.

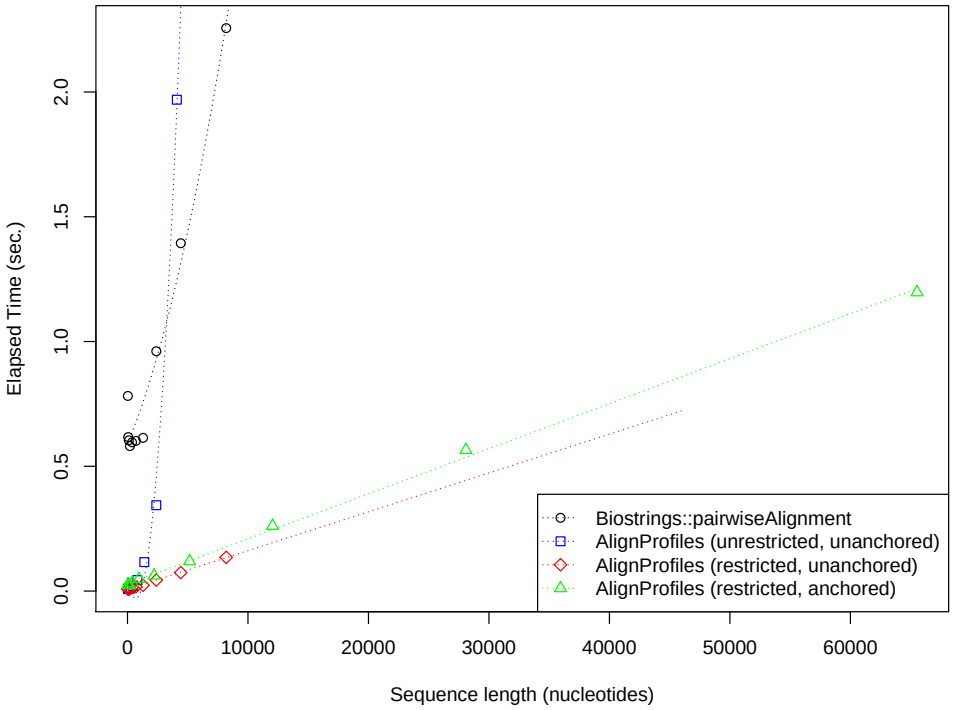

Figure 3: Global Pairwise Sequence Alignment Timings.

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 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 and other sequence alignment software on structural amino acid benchmarks [2]. All structural 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 ClustalW [7]. Most importantly, the accuracy of amino acid alignment begins to drop-off when sequences in the reference alignment have less than 40% average pairwise identity. 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 reverse translates to obtain aligned DNA/RNA sequences.

4 Example: Single Gene Alignment

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

```
> 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
  A DNAStringSet instance of length 317
      width seq                                               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 or their translations (amino acid sequences). Both methods result in an aligned set of DNA sequences, unless the argument *asAAStringSet* is `TRUE` in `AlignTranslation`. A quick inspection reveals that the method of translating before alignment yields a more appealing result. However, if the dna did not belong to a coding sequence then the only option would be to use `AlignSeqs`.

```
> AA <- AlignTranslation(dna, asAAStringSet=TRUE) # align the translation
> BrowseSequences(AA, highlight=1) # view the alignment
> DNA <- AlignTranslation(dna) # align the translation then reverse translate
> DNA <- AlignSeqs(dna) # align the sequences directly without translation
```

5 Example: Building a Guide Tree

The `AlignSeqs` function uses a guide tree to decide which order to align pairs of sequence profiles. The *guideTree* input is a *data.frame* with the grouping of each sequence at increasing levels of dissimilarity

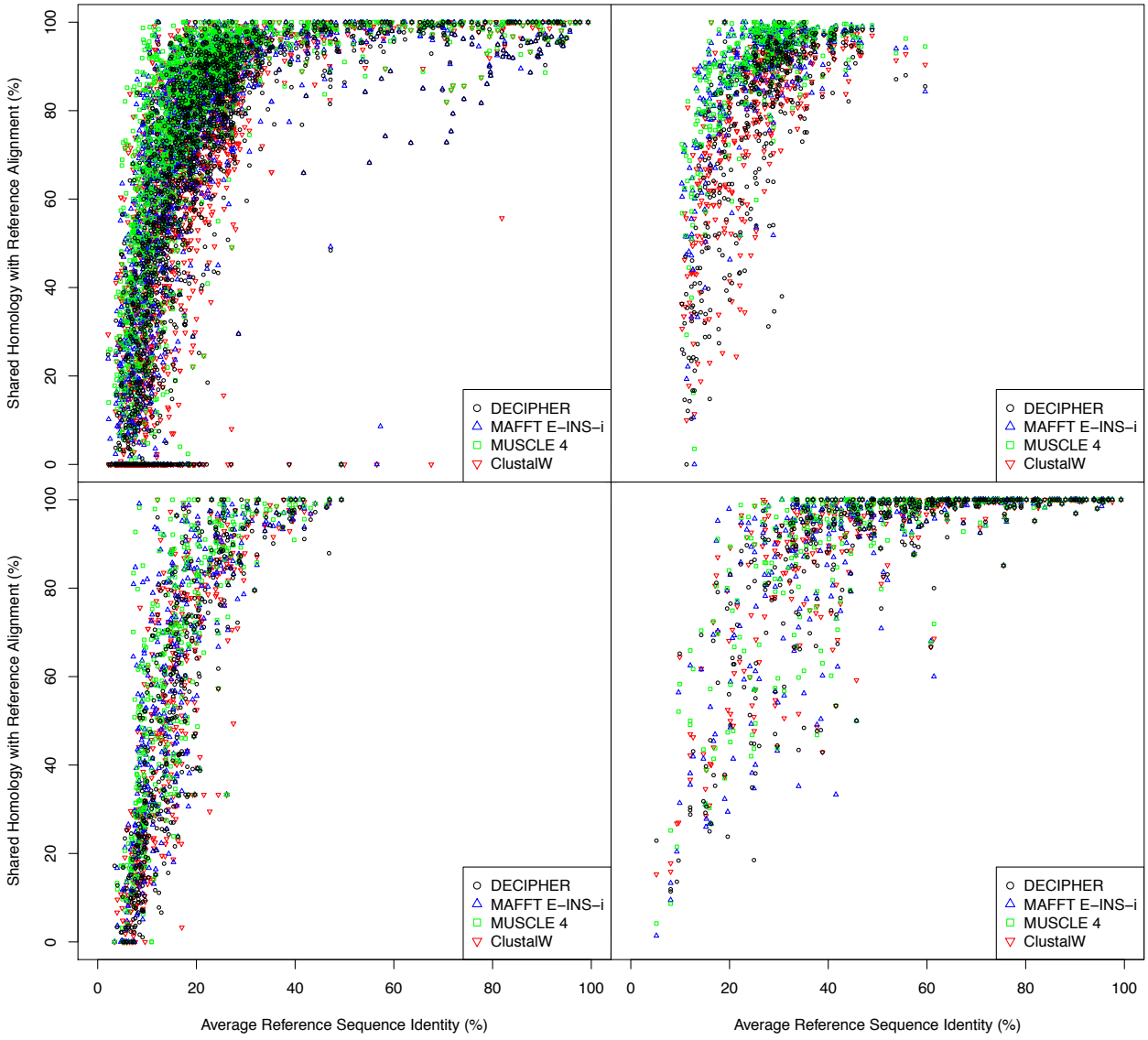

Figure 4: Performance comparison between different programs for multiple alignment ([7], [3], [5]) using amino acid structural benchmarks. The x-axis shows percent identity between sequences in each reference alignment. The y-axis gives the percentage of correctly aligned residues in the estimated alignment according to the reference alignment (SP-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 [8]. The lower-left plot is for SABMARK (version 1.65) [9]. The lower-right plot gives the results on the OXBENCH alignments [6].

between the groups. By default this guide tree is generated directly from the sequences using the order of shared k-mers (i.e., when the argument *guideTree* is `NULL`). However, in some circumstances it may be desirable to provide a guide tree as input.

The first circumstance in which a guide tree could be useful is when the sequences are highly divergent. In such a case the quick guide tree generated by `AlignSeqs` may perform worse than a more accurate guide tree generated from pairwise alignments.

```
> # form guide tree using pairwiseAlignment
> l <- length(dna)
> d <- matrix(0, nrow=l, ncol=l)
> pBar <- txtProgressBar(style=3)
> for (j in 2:l) {
        d[j, 1:(j - 1)] <- pairwiseAlignment(rep(dna[j], j - 1),
                dna[1:(j - 1)],
                scoreOnly=TRUE)
        setTxtProgressBar(pBar, j/l)
 }
> close(pBar)
> # rescale the distance scores from 0 to 1
> m <- max(d[lower.tri(d)])
> d[lower.tri(d)] <- d[lower.tri(d)] - m
> m <- min(d[lower.tri(d)])
> d[lower.tri(d)] <- d[lower.tri(d)]/m
> # form a guide tree from the distance matrix
> gT <- IdClusters(d, cutoff=seq(0, 1, 0.001))
> # use the guide tree as input for alignment
> DNA <- AlignSeqs(dna, guideTree=gT) # align directly
> DNA <- AlignTranslation(dna, guideTree=gT) # align by translation
```

A second circumstance is when there are a large number of sequences (thousands), in which case a faster guide tree may be necessary or desired. For example, beyond 46,340 sequences an alternative guide tree is always required because this is the largest guide tree that can be constructed by DECIPHER. In this case a rough guide tree can be generated by directly clustering the sequences.

```
> # form guide tree using inexact clustering
> gT <- IdClusters(myXStringSet=dna, method="inexact", cutoff=seq(0.05, 0.9, 0.05))
> # use the guide tree as input for alignment
> DNA <- AlignSeqs(dna, guideTree=gT) # align directly
> DNA <- AlignTranslation(dna, guideTree=gT) # align by translation
```

When aligning thousands of amino acid sequences, a chained guide tree may perform well for matching similar residues in the proteins' structures [1]. With a chained guide tree, sequences are added one-by-one to a growing profile representing all of the aligned sequences. Figure 5 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 requires negligible time to construct and often gives an alignment of preferable quality when thousands of protein sequences are being aligned. A chained guide tree can easily be generated, as shown below.

```
> # form a chained guide tree
> gT <- data.frame(seq_along(dna))
> # use the guide tree as input for alignment
> DNA <- AlignTranslation(dna, guideTree=gT) # align by translation
```

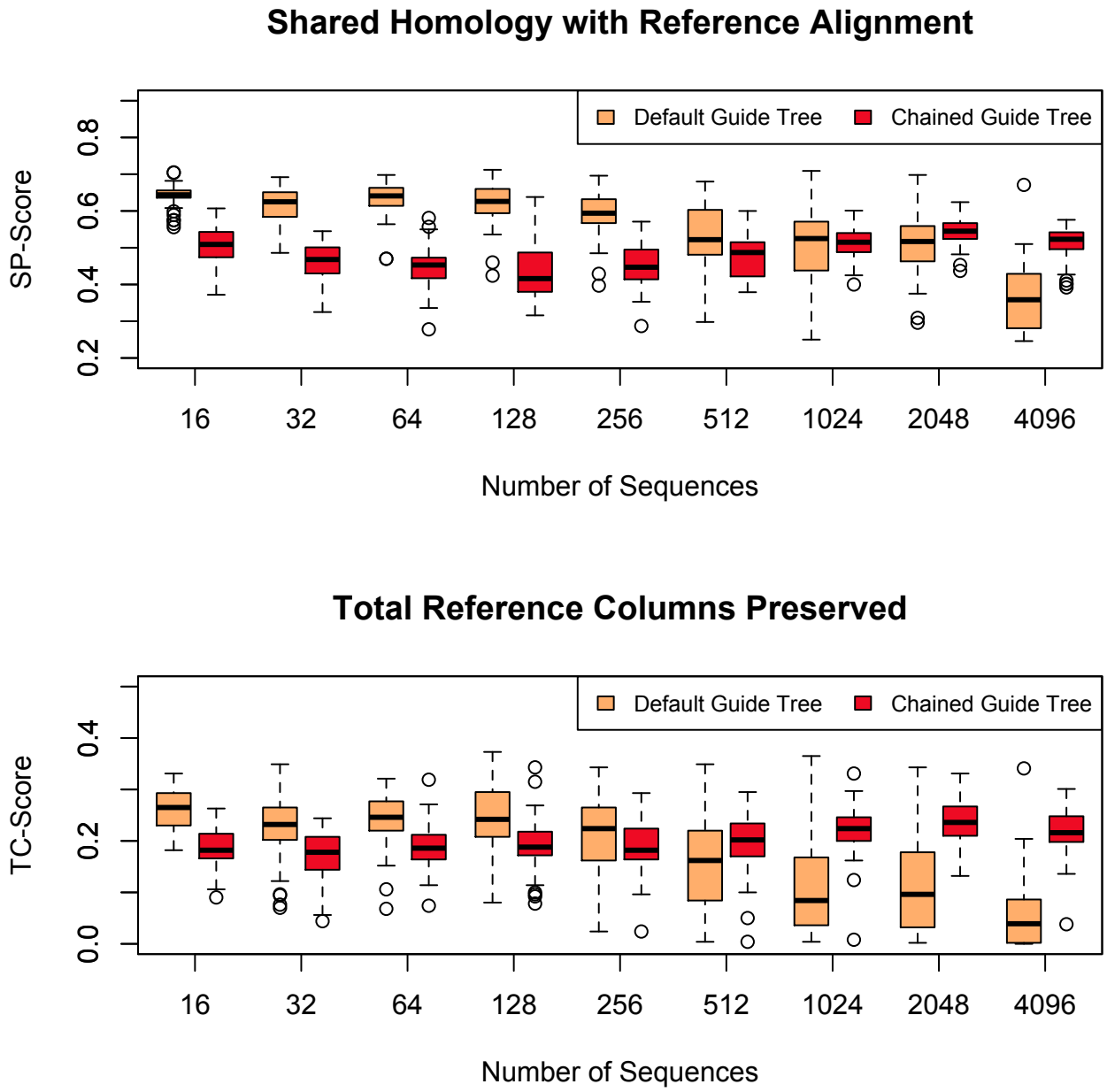

Figure 5: Performance comparison between the default and chained guide trees when aligning increasing numbers of Cytochrome P450 sequence sets. Use of a chained guide tree often provides a better alignment than the default guide tree when aligning thousands of protein sequences.

6 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, asAAStringSet=TRUE)
> AA2 <- AlignTranslation(dna2, asAAStringSet=TRUE)
> AA <- AlignProfiles(AA1, AA2)
```

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(SQLite(), ":memory:")
> Seqs2DB(AA1, "DNAStringSet", dbConn, "AA1", tblName="AA1")
> Seqs2DB(AA2, "DNAStringSet", dbConn, "AA2", tblName="AA2")
> AlignDB(dbConn, tblName=c("AA1", "AA2"), add2tbl="AA",
          gapExtension=-5, gapOpening=-7, terminalGap=-5,
          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.

7 Session Information

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

- R version 3.1.1 Patched (2014-09-25 r66681), `x86_64-apple-darwin13.1.0`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: BiocGenerics 0.12.0, Biostrings 2.34.0, DBI 0.3.1, DECIPHER 1.12.0, IRanges 2.0.0, RSQLite 0.11.4, S4Vectors 0.4.0, XVector 0.6.0
- Loaded via a namespace (and not attached): tools 3.1.1, zlibbioc 1.12.0

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] 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.

[7] 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.

[8] 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.

[9] 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.