

BioTools: Tools based on Biostrings (alignment, classification, database)

Michael Hahsler
Southern Methodist University

Anurag Nagar
Southern Methodist University

Abstract

There are many stand-alone tools available for Bioinformatics. This package aims at using R and the Biostrings package as the common interface for several important tools for multiple sequence alignment (*clustalw*, *kalign*), classification (RDP), sequence retrieval (BLAST) as well as database driven sequence management for 16S rRNA.

Keywords: bioinformatics, Bioconductor, biostrings, sequence alignment, sequence classification, sequence management.

1. Introduction

There are many tools available for sequence alignment and classification. Some tools are: BALibase ([Smith and Waterman 1981](#)), BLAST ([Altschul, Gish, Miller, Myers, and Lipman 1990](#)), T-Coffee ([Notredame, Higgins, and Heringa 2000](#)), MAFFT ([Katoh, Misawa, Kuma, and Miyata 2002](#)), MUSCLE ([Edgar 2004b,a](#)), Kalign ([Lassmann and Sonnhammer 2006](#)) and ClustalW2 and ClustalX2 ([Larkin, Blackshields, Brown, Chenna, McGettigan, McWilliam, Valentin, Wallace, Wilm, Lopez, Thompson, Gibson, and Higgins 2007](#)). Typically, these tools have a command-line interface and the input and output data is stored in files using various formats. Also the parameters supplied to the command-line interface are different. All this makes using and comparing several approaches time consuming and error prone. The R-based Bioconductor project ([Gentleman, Carey, Bates, and others 2004](#)) provides important infrastructure to handle and manipulate bioinformatics data. The **Biostrings** package in particular provides infrastructure for DNA, RNA and protein sequences as well as (multiple) alignments. Also algorithms for sequence alignment are included. However, for multiple sequence alignment using BLAST the user needs to export the data into a file and then run the needed tool manually and re-import the results. Also, **Biostrings** stores sets of sequences in memory and does not directly support storing and querying classification information.

In **BioTools** we provide a simple interface to a growing set of popular tools. The tools are called directly from within R and no manual data export or import is needed. Currently we interface *clustalw*, *kalign*, *RDP* and *clustalw*. **BioTools** also provides database backed sequence management where large amounts of sequences and classification information can be stored and used for selective and efficient sequence retrieval.

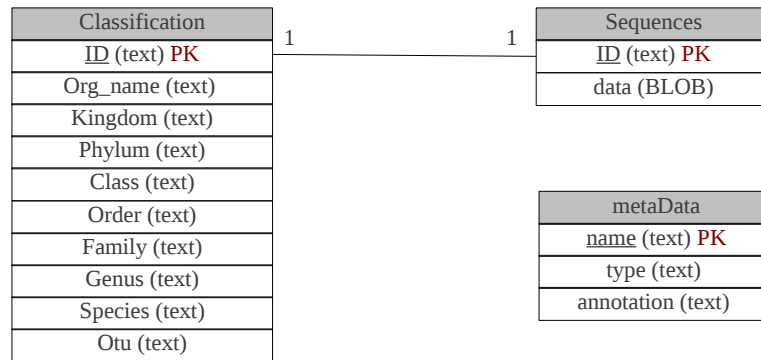


Figure 1: Entity Relationship diagram of GenDB

2. Needed Software

Database comes with R

BioTools_Software_Wizard tries to download and install the needed software (RDP, clustal, kalign, MAFFT, BLAST and boxshade).

3. GenDB: Sequence storage and management

BioTools provides a databases (GenDB) which can be used for efficient storage and retrieval of genetic sequences. By default the light-weight SQLite database is used, but any other compatible database such as MySQL or Oracle can also be used. Figure 1 shows the basic table layout of a GenDB instance with a table containing classification information, a table containing the sequence information and a meta data table. Each sequence we will have an entry in the classification table and an corresponding entry in the sequence table. The tables are connected by a unique sequence ID as the primary key.

GenDB is easy to use. First, we load the library into the R environment.

```
R> library(BioTools)
```

To start we need to create an empty GenDB to store and organize sequences.

```
R> db<-createGenDB("example.sqlite")
R> db
```

```
Object of class GenDB with 0 sequences
DB File: example.sqlite
Tables: sequences
```

The above command creates an empty database with a table structure similar to Figure 1 and stores it in the file example.sqlite. If a GenDB already exists, then it can be opened using `openGenDB()`.

The next step is to import sequences into the database by reading FASTA files. This is accomplished by function `addSequences()`. This function automatically extracts the classification information from the FASTA file's description lines. The default is to expect classification in the format used by the Greengenes project, however other meta data readers can be implemented (see manual page for `addSequences`).

The command below uses a FASTA file provided by the package, hence we use `system.file()` instead of just a string with the file name.

```
R> addSequences(db,
+   system.file("examples/Firmicutes.fasta", package="BioTools"))
```

```
Read 100 sequences. Added 100 sequences.
```

After inserting the sequences, various querying and limiting functions can be used to check the data and obtain a subset of the sequences. To get a count of the number of sequences in the database, the function `nSequences()` can be used.

```
R> nSequences(db)
```

```
[1] 100
```

The function `getSequences()` returns the sequences as a vector. In the following example we get all sequences in the database and then show the first 50 bases of the first sequence.

```
R> s <- getSequences(db)
R> s
```

```
A DNAStringSet instance of length 100
```

	width	seq	names
[1]	1521	TTTGATCCTGGCTCAGG...CGGCTGGATCACCTCCT	1250
[2]	1392	ACGGGTGAGTAACGCGT...TTGGGGTGAAGTCGTAA	13651
[3]	1384	TAGTGGCGGACGGGTGA...TCGAATTTGGGTCAAGT	13652
[4]	1672	GGCGTGCCTAACACATG...TGTAACACGACTTCAT	13654
[5]	1386	ATCTCACCTCTCAATAG...CGAAGGTGGGGTTGGTG	13655
...	
[96]	1446	ATGCAAGTCGAACGGGG...GGGGCCGATGATTGGGG	13857
[97]	1511	ATCCTGGCTCAGGACGA...AGTCGTAACAAGGTAGC	13858
[98]	1544	ATCCTGGCTCAGGACGA...GGTGGATCACCTCCTTC	13860
[99]	1482	GGACGAACGCTGGCGGC...GCCGATGATTGGGGTGA	13861
[100]	1485	GACGAACGCTGGCGGCG...GAAGTCGTAACAAGGTA	13862

```
R> length(s)
```

```
[1] 100
```

```
R> s[[1]]
```

```
1521-letter "DNASTring" instance
seq: TTTGATCCTGGCTCAGGACGAACGCTGGCGG...TGTACCGGAAGGTGCGGCTGGATCACCTCCT
```

```
R> substr(s[[1]], 1, 50)
```

```
50-letter "DNASTring" instance
seq: TTTGATCCTGGCTCAGGACGAACGCTGGCGGCGTGCCTAATGCATGCAAG
```

Sequences in the database can also be filtered using classification information. For example, we can get all sequences of the genus name “Desulfosporomusa” by specifying rank and name.

```
R> s <- getSequences(db, rank="Genus", name="Desulfosporomusa")
R> s
```

```
A DNASTringSet instance of length 7
      width seq                      names
[1]  1498 TNGAGAGTTTGATCCTGG...TGGGGCCGATGATCGGGG 13834
[2]  1481 CTGGCGGCGTGCCTAACA...ATTGGGGTGAAGTCGTAA 13836
[3]  1510 GACGAACGCTGGCGGCGT...AGCCGTATCGGAAGGTGC 13839
[4]  1503 ACGCTGGCGGCGTGCCTA...GGTAGCCGTATCGGAAGG 13844
[5]  1503 ACGCTGGCGGCGTGCCTA...GGTAGCCGTATCGGAAGG 13845
[6]  1429 ACGCTGGCGGCGTGCCTA...GAAGCCGGTGGGGTAACC 13846
[7]  1504 ACGCTGGCGGCGTGCCTA...GGTAGCCGTATCGGAAGG 13847
```

To obtain a single sequence, `getSequences` can be used with rank equal to “id” and supplying the sequence’s greengenes ID as the name.

```
R> s <- getSequences(db, rank="id", name="1250")
R> s
```

```
A DNASTringSet instance of length 1
      width seq                      names
[1]  1521 TTTGATCCTGGCTCAGGA...GCGGCTGGATCACCTCCT 1250
```

The database also stores a classification hierarchy. We can obtain the classification hierarchy used in the database with `getTaxonomyNames()`.

```
R> getTaxonomyNames(db)

[1] "Kingdom" "Phylum" "Class" "Order" "Family" "Genus"
[7] "Species" "Otu" "Org_name" "Id"
```

To obtain all unique names stored in the database for a given rank we can use `getRank()`.

```
R> getRank(db, rank="Order")

[1] "Thermoanaerobacterales" "Clostridiales"
```

The 100 sequences in our example data base contain organisms from different orders. We can obtain the rank name for each sequence individually by using `all=TRUE` or count how many sequences we have for each genus using `count=TRUE`.

```
R> getRank(db, rank="Genus", all=TRUE)
```

[1] Coprothermobacter	Desulfotomaculum
[3] Desulfotomaculum	Desulfotomaculum
[5] Desulfotomaculum	Desulfotomaculum
[7] Desulfotomaculum	Desulfotomaculum
[9] Desulfotomaculum	Pelotomaculum
[11] Desulfotomaculum	Desulfotomaculum
[13] Pelotomaculum	Desulfotomaculum
[15] Desulfotomaculum	Desulfotomaculum
[17] Desulfotomaculum	Pelotomaculum
[19] Desulfotomaculum	Desulfotomaculum
[21] Desulfotomaculum	Desulfotomaculum
[23] Desulfotomaculum	Desulfotomaculum
[25] Pelotomaculum	Syntrophomonas
[27] Syntrophomonas	Syntrophomonas
[29] Syntrophomonas	Syntrophomonas
[31] unknown	Syntrophomonas
[33] Moorella	Moorella
[35] Moorella	Moorella
[37] Thermacetogenium	Thermaerobacter
[39] Carboxydotherrmus	Carboxydotherrmus
[41] Thermoanaerobacterium	Thermoanaerobacterium
[43] Thermoanaerobacterium	Thermoanaerobacterium
[45] Thermoanaerobacterium	Thermoanaerobacterium
[47] Thermoanaerobacterium	Thermoanaerobacterium
[49] Thermoanaerobacter	Thermoanaerobacter
[51] Thermoanaerobacter	Thermoanaerobacter
[53] Thermoanaerobacter	Thermoanaerobacter
[55] Thermoanaerobacter	Thermoanaerobacter
[57] Thermoanaerobacter	Thermoanaerobacter
[59] Selenomonas	Selenomonas
[61] Selenomonas	Selenomonas
[63] Selenomonas	Mitsuokella
[65] Selenomonas	Selenomonas
[67] Selenomonas	unknown
[69] Selenomonas	Veillonella
[71] Veillonella	Veillonella
[73] Veillonella	Veillonella
[75] Dialister	Dialister
[77] Dialister	Desulfosporomusa
[79] Desulfosporomusa	unknown
[81] unknown	Desulfosporomusa

```

[83] Thermosinus          Thermosinus
[85] unknown              Desulfosporomusa
[87] Desulfosporomusa     Desulfosporomusa
[89] Desulfosporomusa     unknown
[91] unknown              Acidaminococcus
[93] Acidaminococcus      unknown
[95] unknown              unknown
[97] Phascolarctobacterium Phascolarctobacterium
[99] unknown              unknown
19 Levels: Acidaminococcus Carboxydothemus ... Veillonella

```

```
R> getRank(db, rank="Genus", count=TRUE)
```

Desulfotomaculum	unknown	Thermoanaerobacter
20	12	10
Selenomonas	Thermoanaerobacterium	Desulfosporomusa
9	8	7
Syntrophomonas	Veillonella	Moorella
6	5	4
Pelotomaculum	Dialister	Acidaminococcus
4	3	2
Carboxydothemus	Phascolarctobacterium	Thermosinus
2	2	2
Coprothermobacter	Mitsuokella	Thermacetogenium
1	1	1
Thermaerobacter		
1		

This information can be easily turned into a barplot showing the abundance of different orders in the data database (see Figure 3).

```

R> oldpar <- par(mar=c(12,5,5,5)) ### make space for labels
R> barplot(sort(
+   getRank(db, rank="Genus", count=TRUE, removeUnknown=TRUE),
+   decreasing=TRUE), las=2)
R> par(oldpar)

```

Filtering also works for `getRank()`. For example, we can find the genera within the order “Thermoanaerobacterales”.

```

R> getRank(db, rank="Genus",
+   whereRank="Order", whereName="Thermo")

```

```

[1] "Coprothermobacter" "Moorella"          "Thermacetogenium"
[4] "Carboxydothemus"  "Thermoanaerobacter"

```

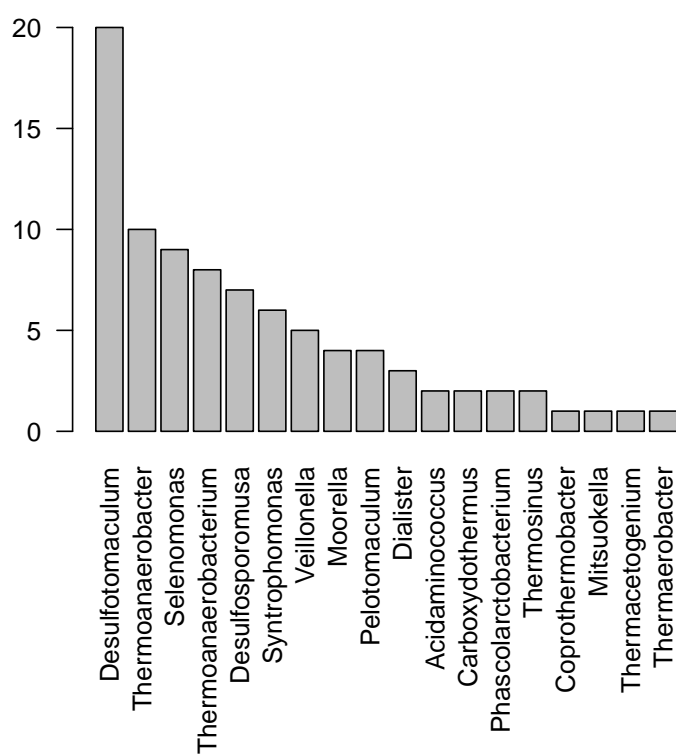


Figure 2: Abundance of different orders in the database.

Note that partial matching is performed from “Thermo” to “Thermoanaerobacterales.” Partial matching is available for ranks and names in most operations in **BioTools**.

We can also get the complete classification hierarchy for different ranks down to individual sequences. In the following we get the classification hierarchy for genus *Thermaerobacter*, then all orders matching *Therm* and then for a list of names.

```
R> getHierarchy(db, rank="Genus", name="Thermaerobacter")
```

Kingdom	Phylum	Class
"Bacteria"	"Firmicutes"	"Clostridia"
Order	Family	Genus
"Clostridiales"	"Sulfobacillaceae"	"Thermaerobacter"
Species	Otu	Org_name
NA	NA	NA
Id		
NA		

```
R> getHierarchy(db, rank="Genus", name="Therm")
```

	Kingdom	Phylum	Class	Order	
[1,]	"Bacteria"	"Firmicutes"	"Clostridia"	"Thermoanaerobacterales"	
[2,]	"Bacteria"	"Firmicutes"	"Clostridia"	"Clostridiales"	
[3,]	"Bacteria"	"Firmicutes"	"Clostridia"	"Clostridiales"	
[4,]	"Bacteria"	"Firmicutes"	"Clostridia"	"Thermoanaerobacterales"	
[5,]	"Bacteria"	"Firmicutes"	"Clostridia"	"Clostridiales"	
	Family				
[1,]	"Thermoanaerobacteraceae"				
[2,]	"Sulfobacillaceae"				
[3,]	"Thermoanaerobacterales Family III. Incertae Sedis"				
[4,]	"Thermoanaerobacteraceae"				
[5,]	"Veillonellaceae"				
	Genus	Species	Otu	Org_name	Id
[1,]	"Thermacetogenium"	NA	NA	NA	NA
[2,]	"Thermaerobacter"	NA	NA	NA	NA
[3,]	"Thermoanaerobacterium"	NA	NA	NA	NA
[4,]	"Thermoanaerobacter"	NA	NA	NA	NA
[5,]	"Thermosinus"	NA	NA	NA	NA

```
R> getHierarchy(db, rank="Genus", name=c("Acid", "Thermo"))
```

	Kingdom	Phylum	Class	Order
[1,]	"Bacteria"	"Firmicutes"	"Clostridia"	"Clostridiales"
[2,]	"Bacteria"	"Firmicutes"	"Clostridia"	"Clostridiales"
[3,]	"Bacteria"	"Firmicutes"	"Clostridia"	"Thermoanaerobacterales"
[4,]	"Bacteria"	"Firmicutes"	"Clostridia"	"Clostridiales"
	Family			


```

[1,] "Veillonellaceae"
[2,] "Thermoanaerobacterales Family III. Incertae Sedis"
[3,] "Thermoanaerobacteraceae"
[4,] "Veillonellaceae"
      Genus      Species Otu Org_name Id
[1,] "Acidaminococcus"   NA   NA   NA   NA
[2,] "Thermoanaerobacterium" NA   NA   NA   NA
[3,] "Thermoanaerobacter"  NA   NA   NA   NA
[4,] "Thermosinus"        NA   NA   NA   NA

```

To get individual sequences we can use again the unique sequence id.

```
R> getHierarchy(db, rank="id", name="1250")
```

```

      Kingdom
      "Bacteria"
      Phylum
      "Firmicutes"
      Class
      "Clostridia"
      Order
      "Thermoanaerobacterales"
      Family
      "Thermodesulfobiaceae"
      Genus
      "Coprothermobacter"
      Species
      "unknown"
      Otu
      "otu_2281"
      Org_name
      "X69335.1Coprothermobacterproteolyticustr.ATCC35245"
      Id
      "1250"

```

Finally, we can close a GenDB after we are done working with it. The database can later be reopened using `openGenDB()`.

```
R> closeGenDB(db)
```

To permanently remove the database we need to delete the file (for SQLite databases) or remove the database using the administrative tool for the database management system.

```
R> unlink("example.sqlite")
```

FIXME: Is there a purge function in DBI to do this?

4. Multiple Sequence Alignment

Multiple Sequence Alignment (MSA) involves comparing and aligning more than two sequences to each other and also possibly to many others in a sequence database. The aim is to discover regions of high similarity for all the sequences taken together. The sequences are generally related such as those from the same species or same phylum.

Although, computationally complex, MSA is quite often what biologists need to identify and characterize sequences from a given group. Sequences might also share an evolutionary relationship, such as having a common ancestor. Such sequences are said to be homologous. Similarly, biologists might be interested in the similarity of genes from different organisms and want to compare their sequences. Another area of application is to find regions which are conserved for a given species or genus. Such conserved regions can be used for identification and classification of organisms.

MSA is a NP-hard problem ?? and is computationally more complex than pairwise alignment. Various algorithms that are used for pairwise alignment, such as dynamic programming, can also be used for MSA but have much greater run time requirements. To obtain results in reasonable time, various heuristics have been proposed such as Progressive Alignment, Iterative Refinement methods, and Hidden Markov Models ?. Out of these, progressive alignment is the most commonly used in many tools for MSA such as Clustal?.

Current methods for Clustal are through an online interface through the The European Bioinformatics Institute website at <http://www.ebi.ac.uk/Tools/msa/clustalw2/> or through a web-service also at the same website. There is no current tool that can be run through the command line for a batch of sequences. Our package addresses this need by providing an interface that can be used for DNA Sequences.

The **BioTools** provides a rich set of functionality for MSA operations including visualization options. The commands below will illustrate that in detail.

4.1. clustalw

```
R> dna <- readDNASTringSet(system.file("examples/DNA_example.fasta",
+   package="BioTools"))
R> dna <- narrow(dna, start=1, end=60)
R> al <- clustal(dna)
R> al
```

DNAMultipleAlignment with 5 rows and 98 columns

	aln	names
[1]	-----...-GTGGCGGACGGGTGAGTAA	4403
[2]	-----...-GTGGCGGACGG-----	4404
[3]	-----...CGTGGCGCA-----	4399
[4]	AGAGTTTGATCCTGGCTCAGA...-----	1675
[5]	AGAGTTTGATTATGGCTCAGA...-----	4411

```
R> ### inspect alignment
R> detail(al)
```

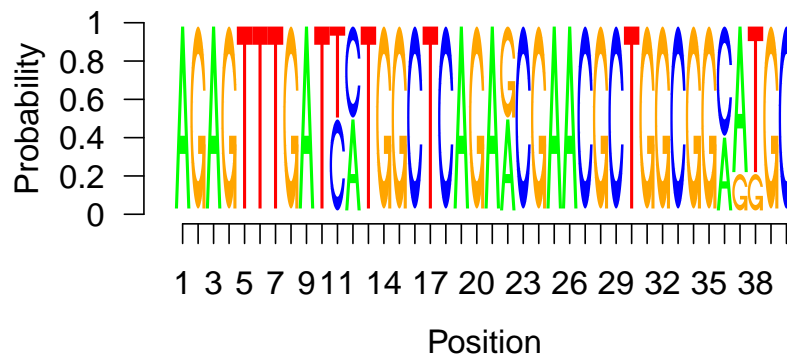


Figure 3: Sequence logo of alignment.

```

4403  --- -----GGAATGCTNAAACACATGCAAGTCGCACGG--
4404  --- -----GCTGGCGGAATGCTTAACACATGCAAGTCGCACGGGG
4399  --- -----GCTGGCGGCAAGCTTAACACATGCAAGTCGAACGGGG
1675  AGA TTTGATCCTGGCTCAGAACGAACGCTGGCGGCCTGCCTAACACATGCAAGTCGAAC---
4411  AGA TTTGATTATGGCTCAGAGCGAACGCTGGCGGCATGCTTAACACATGCAAGTCGCAC---
consensus 1.....10.....20.....30.....40.....50.....
          gctggcGGcatGCTTAACACATGCAAGTCGCACgg

4403  --- GCAGC--AATGTCA-GTGGCGGACGGGTGAGTAA
4404  --- GTTTC--GGCCTTA-GTGGCGGACGG-----
4399  --- ACCTTCGGGTCTTACGTGGCGCA-----
1675  AGA -----
4411  AGA -----
consensus g      aa  t  a  gtggcg  a
          61.....70.....80.....90...

```

Figure 4: Representaion of a DNA multiple alignment using boxshade.

```
R> plot(al, 1, 40)
```

```
R> boxshade(al, file="alignment.pdf")
```

```
R> rna <- readRNAStringSet(system.file("examples/RNA_example.fasta",
+   package="BioTools"))
R> rna
```

```

A RNAStringSet instance of length 5
      width seq                                     names
[1]  1481 AGAGUUUGAUCCUGGCUC...AGUCGUAACAAGGUAACC 1675 AB015560.1 d...
[2]  1404 GCUGGCGGCAGGCCUAAC...UAAGGUCAGCGACUGGGG 4399 D14432.1 Rho...
[3]  1426 GGAAUGCUNAACACAUGC...GGUAGCCGUAGGGGAACC 4403 X72908.1 Ros...
[4]  1362 GCUGGCGGAAUGCUUAAC...UAGGUGUCUAGGCUAACC 4404 AF173825.1 A...
[5]  1458 AGAGUUUGAUUAUGGCUC...UCGUAACAAGGUAACCGU 4411 Y07647.2 Dre...

```

```
R> al <- clustal(rna)
```

```
R> al
```

```
RNAMultipleAlignment with 5 rows and 1500 columns
```

	aln	names
[1]	-----...AAGGUAGCCGUAGGGAACC	4403
[2]	-----...-----	4404
[3]	AGAGUUUGAUUAUGGCUCAGA...AAGGUAACCGU-----	4411
[4]	-----...-----	4399
[5]	AGAGUUUGAUCCUGGCUCAGA...AAGGUAACC-----	1675

```
R> aa <- readAAStringSet(system.file("examples/Protein_example.fasta",
+                                     package="BioTools"))
```

```
R> aa
```

```
A AAStringSet instance of length 5
```

	width	seq	names
[1]	170	MKKSRRRIWIFGLLFSIW...DVYYLEAPFFQGRKCGGT	gi 340754543 ref ...
[2]	233	MYIIWKLLFFKGENVVEH...KEEEVISVDDILKKRRE	gi 340754544 ref ...
[3]	326	MKRSLSGIQPSGILHLGN...KKVQEAKEIVGLLGNIYR	gi 340754545 ref ...
[4]	317	MKYYSGVDLGGTNTKIGL...VLGNEAGILGAAALFMLS	gi 340754546 ref ...
[5]	337	MKKMGIILGALVLAAGLV...IVLVPSIGIDKENVAEYK	gi 340754547 ref ...

```
R> al <- clustal(aa)
```

```
R> al
```

```
AAMultipleAlignment with 5 rows and 358 columns
```

	aln	names
[1]	---MKKSRRRIWIFGLLFSIW...-----	gi 340754543 ref ...
[2]	---MYIIWKLLFFKGENVVEH...-----	gi 340754544 ref ...
[3]	MKKMGIILGALVLAAGLVGCG...DKENVAEYK-----	gi 340754547 ref ...
[4]	---MKRSLSGIQPSGILHLGN...ASKKVQEAKEIVGLLGNIYR	gi 340754545 ref ...
[5]	----MKYYSGVDLGGTNTKIG...-----	gi 340754546 ref ...

4.2. kalign

Another popular technique for MSA is based on the KAlign algorithm [Lassmann and Sonnhammer \(2005\)](#). It uses a progressive method for sequence alignment by first calculating pairwise distances between sequences and then constructing a guide tree from these pairwise alignments. The guide tree is used to progressively create the multiple sequence alignment profile. KAlign uses the Wu-Manber approximate string matching algorithm [Wu and Manber \(1992\)](#) for distance calculation. KAlign has been evaluated to be faster and more efficient than other methods [Lassmann and Sonnhammer \(2005\)](#) due to the use of the approximate string matching algorithm and efficient guide tree generation.

```
R> dna <- readDNAStrngSet(system.file("examples/DNA_example.fasta",
+                                     package="BioTools"))
```

```
R> dna
```

```

A DNAStringSet instance of length 5
      width seq                      names
[1]  1481 AGAGTTTGCCTGGCTC...AGTCGTAACAAGGTAACC 1675 AB015560.1 d...
[2]  1404 GCTGGCGGCAGGCCTAAC...TAAGGTCAGCGACTGGGG 4399 D14432.1 Rho...
[3]  1426 GGAATGCTNAACACATGC...GGTAGCCGTAGGGGAACC 4403 X72908.1 Ros...
[4]  1362 GCTGGCGGAATGCTTAAC...TAGGTGTCTAGGCTAACC 4404 AF173825.1 A...
[5]  1458 AGAGTTTGATTATGGCTC...TCGTAACAAGGTAACCGT 4411 Y07647.2 Dre...

R> ### align the sequences
R> al <- kalign(dna)
R> al

```

```

DNAMultipleAlignment with 5 rows and 1502 columns

```

```

      aln                      names
[1] AGAGTTTGCCTGGCTCAGA...-----CAAGGTAAC--C 1675 AB015560.1 d...
[2] G-----...-----TGGG-----G 4399 D14432.1 Rho...
[3] G-----...GGTAGCCGTAGGGGAAC--C 4403 X72908.1 Ros...
[4] G-----...-----TAGGCTAAC--C 4404 AF173825.1 A...
[5] AGAGTTTGATTATGGCTCAGA...-----CAAGGTAACCGT 4411 Y07647.2 Dre...

```

4.3. MUSCLE

```

R> dna <- readDNAStringSet(system.file("examples/DNA_example.fasta",
+   package="BioTools"))
R> dna

```

```

A DNAStringSet instance of length 5
      width seq                      names
[1]  1481 AGAGTTTGCCTGGCTC...AGTCGTAACAAGGTAACC 1675 AB015560.1 d...
[2]  1404 GCTGGCGGCAGGCCTAAC...TAAGGTCAGCGACTGGGG 4399 D14432.1 Rho...
[3]  1426 GGAATGCTNAACACATGC...GGTAGCCGTAGGGGAACC 4403 X72908.1 Ros...
[4]  1362 GCTGGCGGAATGCTTAAC...TAGGTGTCTAGGCTAACC 4404 AF173825.1 A...
[5]  1458 AGAGTTTGATTATGGCTC...TCGTAACAAGGTAACCGT 4411 Y07647.2 Dre...

```

```

R> al <- MUSCLE(dna)
R> al

```

```

DNAMultipleAlignment with 5 rows and 1502 columns

```

```

      aln                      names
[1] AGAGTTTGCCTGGCTCAGA...AAGGTAACC----- 1675
[2] -----...----- 4399
[3] AGAGTTTGATTATGGCTCAGA...AAGGTAACCGT----- 4411
[4] -----...AAGGTAGCCGTAGGGGAACC 4403
[5] -----...----- 4404

```

```
R> ### inspect alignment
R> detail(al)
```

4.4. MAFFT

```
R> dna <- readDNASTringSet(system.file("examples/DNA_example.fasta",
+   package="BioTools"))
R> dna
```

```
A DNASTringSet instance of length 5
      width seq                                     names
[1]  1481 AGAGTTTGATCCTGGGCTC...AGTCGTAACAAGGTAACC 1675 AB015560.1 d...
[2]  1404 GCTGGCGGCAGGCCTAAC...TAAGGTCAGCGACTGGGG 4399 D14432.1 Rho...
[3]  1426 GGAATGCTNAACACATGC...GGTAGCCGTAGGGGAACC 4403 X72908.1 Ros...
[4]  1362 GCTGGCGGAATGCTTAAC...TAGGTGTCTAGGCTAACC 4404 AF173825.1 A...
[5]  1458 AGAGTTTGATTATGGCTC...TCGTAACAAGGTAACCGT 4411 Y07647.2 Dre...
```

```
R> al <- mafft(dna)
R> al
```

```
DNAMultipleAlignment with 5 rows and 1499 columns
      aln                                     names
[1] AGAGTTTGATCCTGGGCTCAGA...AAGGTAACC----- 1675
[2] -----...----- 4399
[3] -----...AAGGTAGCCGTAGGGGAACC 4403
[4] -----...----- 4404
[5] AGAGTTTGATTATGGCTCAGA...AAGGTAACCGT----- 4411
```

```
R> ### inspect alignment
R> detail(al)
```

5. Classification with RDP

The Ribosomal Database Project (RDP) provides various tools and services to the scientific community for data related to 16S rRNA sequences. Among other tools, it provides a hierarchical browser and a classifier that can be used to assign sequences to taxonomies. The classifier uses a Naive Bayesian approach to quickly and accurately classify sequences. The classifier uses an alignment-free approach and compares the word frequency distribution with word size of 8 [Wang, Garrity, Tiedje, and Cole \(2007\)](#).

The RDP classifier needs to be trained first before it can be used. The default classifier comes trained with sequences from the microbial 16S rRNA gene.

5.1. Using the default RDP classifier

Use the default classifier

```
R> seq <- readRNAStringSet(system.file("examples/RNA_example.fasta",
+   package="BioTools"))
R> ## shorten names
R> names(seq) <- sapply(strsplit(names(seq), " "), "[", 1)
R> seq
```

```
A RNAStringSet instance of length 5
      width seq                                     names
[1]  1481 AGAGUUUGAUCCUGGCUC...AGUCGUAACAAGGUAACC 1675
[2]  1404 GCUGGCGGCAGGCCUAAC...UAAGGUCAGCGACUGGGG 4399
[3]  1426 GGAAUGCUNAAACAUGC...GGUAGCCGUAGGGGAACC 4403
[4]  1362 GCUGGCGGAAUGCUUAAC...UAGGUGUCUAGGCUAACC 4404
[5]  1458 AGAGUUUGAUUAUGGCUC...UCGUAACAAGGUAACCGU 4411
```

```
R> ## use rdp for classification
R> predict(RDP(), seq)
```

	norank	domain	phylum	class
1675	Root	Bacteria	Proteobacteria	Deltaproteobacteria
4399	Root	Bacteria	Proteobacteria	Alphaproteobacteria
4403	Root	Bacteria	Proteobacteria	Alphaproteobacteria
4404	Root	Bacteria	Proteobacteria	Alphaproteobacteria
4411	Root	Bacteria	Proteobacteria	Alphaproteobacteria

	order	family	genus
1675	<NA>	<NA>	<NA>
4399	Rhodospirillales	Rhodospirillaceae	Rhodovibrio
4403	Rhodospirillales	Acetobacteraceae	Roseococcus
4404	Rhodospirillales	Acetobacteraceae	Roseococcus
4411	Rhodospirillales	Acetobacteraceae	<NA>

5.2. Training a custom RDP classifier

Train a custom RDP classifier on new data

```
R> trainingSequences <- readDNAStringSet(
+   system.file("examples/trainingSequences.fasta", package="BioTools"))
R> customRDP <- trainRDP(trainingSequences)
R> customRDP
```

RDPClassifier

Location: /home/hahsler/baR/QuasiAlign/pkg/BioTools/Work/vignette/classifier

```
R> testSequences <- readDNAStringSet(
+   system.file("examples/testSequences.fasta", package="BioTools"))
R> predict(customRDP, testSequences)
```

	rootrank	Kingdom	Phylum	Class	Order
13811	Root	Bacteria	Firmicutes	Clostridia	Clostridiales
13813	Root	Bacteria	Firmicutes	Clostridia	Clostridiales
13678	Root	Bacteria	Firmicutes	Clostridia	Clostridiales
13755	Root	Bacteria	Firmicutes	Clostridia	Clostridiales
13661	Root	Bacteria	Firmicutes	Clostridia	Clostridiales
					Family
13811					Veillonellaceae
13813					Veillonellaceae
13678					Peptococcaceae
13755	Thermoanaerobacterales	Family III.	Incertae	Sedis	
13661					Peptococcaceae
					Genus
13811		Selenomonas			
13813		Selenomonas			
13678		Desulfotomaculum			
13755	Thermoanaerobacterium				
13661		Desulfotomaculum			

```
R> ## clean up
R> removeRDP(customRDP)
```

6. Sequence Retrieval with BLAST

```
R> seq <- readRNAStringSet(system.file("examples/RNA_example.fasta",
+   package="BioTools"))
R> ## shorten names
R> names(seq) <- sapply(strsplit(names(seq), " "), "[", 1)
R> seq
```

```
A RNAStringSet instance of length 5
      width seq                                names
[1]  1481 AGAGUUUGAUCCUGGCUC...AGUCGUAACAAGGUAACC 1675
[2]  1404 GCUGGCGGCAGGCCUAAC...UAAGGUCAGCGACUGGGG 4399
[3]  1426 GGAAUGCUNAACAACAUAGC...GGUAGCCGUAGGGGAACC 4403
[4]  1362 GCUGGCGGAUAGCUUAAC...UAGGUGUCUAGGCUAACC 4404
[5]  1458 AGAGUUUGAUUAUGGCUC...UCGUAACAAGGUAACCGU 4411
```

```
R> ## load a BLAST database (replace db with the location + name of the BLAST DB)
R> blast <- BLAST(db="~/tmp/blast/16SMicrobial")
R> blast
```

```
BLAST Database
Location: /home/hahsler/tmp/blast/16SMicrobial
```



```
R> print(blast, info=TRUE)
```

```
BLAST Database
```

```
Location: /home/hahsler/tmp/blast/16SMicrobial
```

```
Database: 16S Microbial Sequences
```

```
      8,412 sequences; 12,354,954 total bases
```

```
Date: Mar 26, 2013 12:51 AM
```

```
Longest sequence: 1,768 bases
```

```
Volumes:
```

```
      /home/hahsler/tmp/blast/16SMicrobial
```

```
R> ## query a sequence using BLAST
```

```
R> cl <- predict(blast, seq[1,])
```

```
R> cl[1:5,]
```

	QueryID	SubjectID	Perc.Ident	Alignment.Length
1	1675 gi 444304125 ref NR_074549.1		85.99	1249
2	1675 gi 444304125 ref NR_074549.1		94.20	69
3	1675 gi 343198971 ref NR_044205.1		84.40	1314
4	1675 gi 265678428 ref NR_028730.1		82.53	1494
5	1675 gi 343201138 ref NR_041853.1		82.30	1531

	Mismatches	Gap.Openings	Q.start	Q.end	S.start	S.end	E	Bits
1	158	15	235	1478	247	1483	0e+00	1321
2	4	0	1	69	1	69	2e-22	106
3	188	15	87	1392	61	1365	0e+00	1275
4	206	34	31	1475	1	1488	0e+00	1271
5	210	40	3	1481	1	1522	0e+00	1269

7. Creating Random Sequences

Creating random sequences given letter probabilities.

```
R> seqs <- random_sequences(100, number=10, prob=c(a=.5, c=.3, g=.1, t=.1))
```

```
R> seqs
```

```
A DNAStringSet instance of length 10
```

	width	seq	names
[1]	100	CCCGCAACCCCATAGAAA...AGAAAGATAAACAAACA	1
[2]	100	CAAAAAAACATAATTAA...TAGCACCTAGGGGCTCC	2
[3]	100	CACCCAAATCAACCTCCA...CAAACGCATACCCACAA	3
[4]	100	TCATAATCCTCAAAAAAA...AACATTCCCCATCCAAC	4
[5]	100	ACCCACACACGTAGACCA...AACCCACCTACACACCC	5
[6]	100	GGACGCGACATTCACCAC...AAATTCTGACACCCCAA	6
[7]	100	AACAAGACAAGAATAACC...GAGACAGAACAAACACA	7

```
[8] 100 CCAAAACACCTTAAAAAT...ACGACACACCCACGAGA 8
[9] 100 GCAACAACACATCAAAGA...CTAAATCCAAACCTGC 9
[10] 100 ATATAAACAAAAAAATT...TAATAAACTACACATAG 10
```

Creating random sequences using dinucleotide transition probabilities

```
R> prob <- matrix(runif(16), nrow=4, ncol=4, dimnames=list(DNA_BASES, DNA_BASES))
R> prob <- prob/rowSums(prob)
R> seqs <- random_sequences(100, number=10, prob=prob)
R> seqs
```

```
A DNASTringSet instance of length 10
      width seq                                     names
[1] 100 CCGGGGCCCTTAGGGTCGA...GGGGGGGGATTCTGGTT 1
[2] 100 TTCTTCGGGAGTCGAGGA...AGAGGCGTAATCGGTTTC 2
[3] 100 CGGGCCCCCTCAGGCCA...GTCTACCTATATTCTAT 3
[4] 100 AAGGTAAGGGGGGAGGG...ATTTAAGGGGGAAGCG 4
[5] 100 GGGCGTAGATAGAGTCTA...ATAACGACTATAGAGGG 5
[6] 100 TCTTCCTCGCTAGTCCCT...TAGATCAGGAAGGGGGA 6
[7] 100 TCCGAAGTAGGCCCGGG...GGAGGACCTCTATCTAG 7
[8] 100 GAGGGATCTCCCGATTA...GAGGGGAGGCGAATAGG 8
[9] 100 AGCCCTCGTCCTCTCACT...GATTCCGTACGGGGGAT 9
[10] 100 GGACAGGTCCTTTAGGTA...GGATCGAGTCTTTCTCT 10
```

Creates a set of sequences which are random mutations (with base changes, insertions and deletions) for a given DNA, RNA or AA sequence.

```
R> s <- random_sequences(100, number=1)
R> s
```

```
A DNASTringSet instance of length 1
      width seq                                     names
[1] 100 GGCTTTAATCCGAGGCCA...CCTGTGGGGTGGGCACTG 1
```

```
R> ### create 10 sequences with 1 percent base changes, insertions and deletions
R> m <- mutations(s, 10, change=0.01, insertion=0.01, deletion=0.01)
R> m
```

```
A DNASTringSet instance of length 10
      width seq                                     names
[1] 100 GGCTTTAATCCGAGGCCA...TGTGGGGTGCGGCACTG 1_mutation_1
[2] 101 GGCTTTAATCCGAGGCCA...TGTGGGGTGCGGCACTG 1_mutation_2
[3] 100 GGCTTTATCCGAGGCCA...CTGTGGGGTGGGCACTG 1_mutation_3
[4] 101 GGCTTTAATCCGAGGCCA...CTGTGGGGTGGGCACTG 1_mutation_4
[5] 102 GGCTTTAATCCGAGGCCA...CTGTGGGGTGGGCACTG 1_mutation_5
[6] 100 GGCTTTAATCCGAGGCCA...CTGTGGGGTGGGCACTG 1_mutation_6
[7] 101 GGCTTTAATCCGAGGCCA...CCTGTGGGGTGGGCATG 1_mutation_7
```

```
[8] 100 GGCTTTAATCCGAGGCCA...CCTGTGGGGTGGCACTG 1_mutation_8
[9] 99 GGCTTTAATCCGAGGCCAC...CCTGTGGGGTGGCAAT 1_mutation_9
[10] 100 GGCTTTAATCCGAGGCCA...CTGTGGGGTGGCACTT 1_mutation_10
```

```
R> clustal(c(s,m))
```

```
DNAMultipleAlignment with 11 rows and 109 columns
```

	aln	names
[1]	GGCTTTAATCCGAGGCCACC...ACCTGTGGGGTGCGGCACTG	1_mutation_1
[2]	GGCTTTAATCCGAGGCCACC...ACCTGTGGGGTGCGGCACTG	1_mutation_2
[3]	GGCTTTA-TCCGAGGCCACC...ACCTGTGGGGTG-GGCACTG	1_mutation_3
[4]	GGCTTTAATCCGAGGCCACC...ACCTGTGGGGTG-GGCACTG	1_mutation_5
[5]	GGCTTTAATCCGAGGCCACC...ACCTGTGGGGTG-GGCACTG	1
[6]	GGCTTTAATCCGAGGCCACC...ACCTGTGGGGTG-GGCACTG	1_mutation_4
[7]	GGCTTTAATCCGAGGCCACC...ACCTGTGGGGTG-G-CACTG	1_mutation_8
[8]	GGCTTTAATCCGAGGCCACC...ACCTGTGGGGTG-GGCACTG	1_mutation_6
[9]	GGCTTTAA-CCGAGGCCACC...ACCTGTGGGGTG-GGCAAT-	1_mutation_9
[10]	GGCTTTAATCCGAGGCCACC...ACCTGTGGGGTG-GGCATG-	1_mutation_7
[11]	GGCTTTAATCCGAGGCCACC...ACCTGTGGGGTG-GGCACTT	1_mutation_10

8. Calculating Distances between Sequences

Calculating distances between sequences is important for many bioinformatics applications. The following distance metrics are available in **BioTools**:

- Feature frequency profile (distFFP): A FFP is the normalized (by the number of k-mers in the sequence) count of each possible k-mer in a sequence. The distance is defined as the Jensen-Shannon divergence (JSD) between FFPs (Sims and Kim, 2011).
- Composition Vector (distCV): A CV is a vector with the frequencies of each k-mer in the sequence minus the expected frequency of random background nice obtained from a Markov Model (not implemented yet!). The cosine distance is used between CVs. (Qi et al, 2007).
- Numerical Summarization Vector (distNSV): An NSV is frequency distribution of all possible k-mers in a sequence. The Manhattan distance is used between NSVs (Nagar and Hahsler, 2013).
- Distance between sets of k-mers (distkMer): Each sequence is represented as a set of k-mers. The Jaccard (binary) distance is used between sets (number of unique shared k-mers over the total number of unique k-mers in both sequences).
- Distance based on SimRank (distSimRank): 1-simRank (see simRank).
- Edit (Levenshtein) Distance (distEdit): Edit distance between sequences.
- Distance based on alignment score (distAlignment): see stringDist in Biostrings.

- Evolutionary distances (distApe): see dist.dna in ape.

```
R> s <- mutations(random_sequences(100), 100)
R> s
```

```
A DNAStringSet instance of length 100
      width seq                      names
[1]   103 GCTGTAGTGTGCGCCGAG...GGACTACATTTTAGTGG 1_mutation_1
[2]    99 GCTGTAGGTCGCCAAGT...AGGACTACATTTTGTGG 1_mutation_2
[3]   101 GCTGTAGGTCGCACAAG...GGACTACATTTTAGTGG 1_mutation_3
[4]   102 GCTGTATGTCGCCAAGT...GGACTACATTTTAGTGG 1_mutation_4
[5]    99 GCTGTAGGTGCACAAGT...GGACTACATTTTAGTGG 1_mutation_5
...    ...
[96]   102 GCTGTAGGTCGCCAAG...GACTACATTTTAGTTGG 1_mutation_96
[97]   101 GCTGTAGGTCGCCAAGT...GGACTACATTTTAGTGG 1_mutation_97
[98]   101 GCTGTGGTCGCCAAGTA...GGACTACATTTTAGTGG 1_mutation_98
[99]   101 GCTGTAGGTCGCCAAGT...GGACTACATGTTAGTGG 1_mutation_99
[100]  100 GCATGTAGGTCGCCAGT...GGACTACATTTTAGTGG 1_mutation_100
```

```
R> ### calculate NSV distance
R> dNSV <- distNSV(s)
R> ### relationship with edit distance
R> dEdit <- distEdit(s)
R> df <- data.frame(dNSV=as.vector(dNSV), dEdit=as.vector(dEdit))
R> plot(sapply(df, jitter), cex=.1)
R> ### add lower bound (2*k, for Manhattan distance)
R> abline(0,1/(2*3), col="red", lwd=2)
R> ### add regression line
R> abline(lm(dEdit~dNSV, data=df), col="blue", lwd=2)
R> ### check correlation
R> cor(dNSV,dEdit)
```

```
[1] 0.8336
```

9. Conclusion

Acknowledgments

This research is supported by research grant no. R21HG005912 from the National Human Genome Research Institute (NHGRI / NIH).

References

- Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ (1990). “Basic local alignment search tool.” *Journal of Molecular Biology*, **215**(3), 403–410. ISSN 0022-2836.
- Edgar R (2004a). “MUSCLE: a multiple sequence alignment method with reduced time and space complexity.” *BMC Bioinformatics*, **5**(1), 113+. ISSN 1471-2105.
- Edgar RC (2004b). “Muscle: multiple sequence alignment with high accuracy and high throughput.” *Nucleic Acids Research*, **32**, 1792–1797.
- Gentleman RC, Carey VJ, Bates DM, others (2004). “Bioconductor: Open software development for computational biology and bioinformatics.” *Genome Biology*, **5**, R80. URL <http://genomebiology.com/2004/5/10/R80>.
- Katoh K, Misawa K, Kuma K, Miyata T (2002). “MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform.” *Nucleic Acids Research*, **30**(14), 3059–3066.
- Larkin M, Blackshields G, Brown N, Chenna R, McGettigan P, McWilliam H, Valentin F, Wallace I, Wilm A, Lopez R, Thompson J, Gibson T, Higgins D (2007). “Clustal W and Clustal X version 2.0.” *Bioinformatics*, **23**, 2947–2948. ISSN 1367-4803.
- Lassmann T, Sonnhammer EL (2005). “Kalign—an accurate and fast multiple sequence alignment algorithm.” *BMC bioinformatics*, **6**(1), 298.
- Lassmann T, Sonnhammer EL (2006). “Kalign, Kalignvu and Mumsa: web servers for multiple sequence alignment.” *Nucleic Acids Research*, **34**. ISSN 1362-4962.
- Notredame C, Higgins DG, Heringa J (2000). “T-Coffee: A novel method for fast and accurate multiple sequence alignment.” *Journal of Molecular Biology*, **302**(1), 205–217. ISSN 0022-2836.
- Smith TF, Waterman MS (1981). “Identification of common molecular subsequences.” *Journal of Molecular Biology*, **147**(1), 195–197. ISSN 0022-2836.
- Wang Q, Garrity GM, Tiedje JM, Cole JR (2007). “Naive Bayesian classifier for rapid assignment of rRNA sequences into the new bacterial taxonomy.” *Applied and environmental microbiology*, **73**(16), 5261–5267.
- Wu S, Manber U (1992). “Fast Text Searching Allowing Errors.” *Communications of the ACM*, **35**, 83–91.

Affiliation:

Michael Hahsler
Engineering Management, Information, and Systems
Lyle School of Engineering
Southern Methodist University
P.O. Box 750123
Dallas, TX 75275-0123
E-mail: mhahsler@lyle.smu.edu
URL: <http://lyle.smu.edu/~mhahsler>

Anurag Nagar
Computer Science and Engineering
Lyle School of Engineering
Southern Methodist University
P.O. Box 750122
Dallas, TX 75275-0122
E-mail: anagar@smu.edu