Package “alignmentFilter”

Version: 1.3.0
Date: 2023-09-03
Title: Treating Diverse Problems in Alignment
Author: Qiang Zhang, Xinmei Qin, Yongbin Lu, Pengwei Li, Xiaoyuan Mo, Xiyang Huang
Maintainer: Qiang Zhang (qiangzhang04@126.com)
Depends: R (>= 4.1.2). If earlier R versions were used (e.g. perhaps < 3.5.3), certain function(s) of alignmentFilter may not work properly and give wrong results (silently).
Import: magrittr, Biostrings,utils
Description: 1) mask overly-divergent/ambiguously-aligned segments and very short strings (i.e. short contiguous nucleotide strings separated by gap) in alignment; 2) identify and adjust reverse complementary sequences contained in either aligned or unaligned nucleotide data; 3) delete gappy columns in alignment with independent optional thresholds; 4) sift alignments with length longer than a given threshold; 5) delete short sequences in alignment.
License: GPL-3
Repository: https://github.com/qiangzhang04/alignmentFilter
R topics documented (Note R is case sensitive):

Input data required

Data type: Nucleotide sequences are allowed only. In these sequences, only unambiguous nucleotides (i.e. “A”, “T”, “G” and “C”) will be taken into account such as when calculating sequence similarity implemented in the function “maskSegment”. Other letters other than the four nucleotides in sequences will usually be overlooked but will be regarded as gaps (-) when using the function “degap” of alignmentFilter. For example, if certain sites consist of excessive degenerate nucleotide “N”, they will be treated equally as gaps and these sites (columns) may be deleted when employing the “degap” function.

Data Format: Standard FASTA as illustrated below. *Note 1) No blank space is permitted between the FASTA format symbols “>” and the tip (species) names and within tip names, otherwise unexpected (incorrect) output may be produced without any warnings; 2) The tip names and sequences cannot be in the same lines (you can check this in notepad++ or similar tools), otherwise some functions cannot work (or generate errors) or unexpected (incorrect) output may be produced.

```
>PrimulinaChi@11969.p1
ATGGCTTCTTTCTCTTTGTGTTCTCCCCTTTCATCGCTCCACACCTCAGACTCCTCAAAATCTTTCGTTTCCTTCGGA
>PrimulinaCra@63314.p1
ATGGCTTCTTTCTCTTTGTGTTCTCCCCTTTCATCGCTCCACACCTCAGACTCCTCAAAATCTTTCATTTCCTTCGGA
>PrimulinaLnc@53486.p1
ATGGCTTCTTTCTCTTTGTGTTCTCCCCTTTCATCGCTCCACACCTCGGACTCCTCAAAATCTTTCGTTTCCTTCGGA
>PrimulinaLne@97548.p1
ATGGCTTCTTTCTCTTTGTGTTCTCCCCTTTCATCGCTCCACACCTCAGACTCCTCAAAATCATTCGTTTCCTTCGGA
>PrimulinaMol@55113.p1
ATGGCATCTTTCTCTTTGTGTTCTCCCCTTTCATCGCTCCACACCTCAGACTCCTCAAAATCTTTCGTTTCCTTCGGA
>PrimulinaLon@77302.p1
ATGGCTTCTTTCTCTTTGTGTTCTCCCCTTTCATCGCTCCACACCTCGGGCTCCTCAAAATCTTTCGTTTCCTTCGGA
>PrimulinaPun@29464.p1
ATGGCTTCTTTCTCTTTGTGTTCTCCCCTTTCATCGCTCCACACCTCGGGCTCCTCAAAATCTTTCGTTTCCTTCGGA
>PrimulinaMin@25799.p1
ATGGCTTCTTTCTCTTTGTGTTCTCCCCTTTCATCGTTCCACACCTCGGACTCCTCGAAATCTTTCGTTTCCTTCGGA
>PrimulinaWen@41339.p1
ATGGCTTCTTTCTCTTTGTGTTCTCCCCTTTCATCGCTCCACACCTCGGACTCCTCGAAATCTTTCGTTTCCTTCGGA
>PrimulinaSpi@58890.p1
ATGGCTTCTTTCTCTTTGTGTTCTCCCCTTTCATCGCTCCACGCCTCGGACTCCTCAAAATCTTTCGTTTCCTTCGGA
>PrimulinaOph@9116.p1
ATGGCTTCTTTCTCTTTGTGTTCTCCCCTTTCATCGCTCCACGCCTCGGACTCCTCAAAATCTTTCGTTTCCTTCGGA
>PrimulinaHed@56280.p1
ATGGCTTCTTTCTCTTTGTGTTCTCCCCTTTCATCGCTCCACACCTCGGACTCCTCGAAATCTTTCGTTTCCTTCGGA
>PetrocodFan@24516
ATGGCTTCTTTCTCTTTGTGTTCTCCCCTTTCATCGCTCCAGACCTCAGACTCCTCAAAATCTCTCGTATCCTTCGGA

```

Introductions to functions

alignmentLength: sort alignments according to lengths

Description

Extract individual alignments with aligned length longer or shorter than a given threshold.

Usage

alignmentLength(obj, alignment_name, Len = 200, short_alignments = F)

Arguments

obj	A variable of DNA alignment read in R.
alignment_name	The specified file name of the input DNA alignment.
Len	A user's specified number which the alignments with lengths equal/above or below will be sorted into different folders. The default is 200 (bp).
short_alignments	Logical values TRUE (T) or FALSE (F). If TRUE, output the short alignments to a new folder too.

Details

Sort alignments into alternative folders (i.e. “long_alignments/”, “short_alignments/”).

Value

DNA alignments with length longer or shorter than a given threshold (parameter “Len”) into different folders, respectively.

Examples

```
library(magrittr)
# The following is a data set built in this package alignmentFilter.
obj_names <- dir(system.file("extdata", package = "alignmentFilter"), full.names =
  TRUE) %>% grep(pattern = "fasta$", value = TRUE)
obj = lapply(obj_names, readLines)
# But if your data is saved in such as a folder named mydata under disc “D:”, you can
  set the R working directory to the folder and read your data like this if the names of
  the data files ending with suffix “.fasta”.
setwd("D:/mydata")
obj_names <- list.files(pattern = ".fasta$")
obj <- lapply(obj_names, readLines)
# Run single object. Pay attention to that the class (type) of obj is list, so single object
  (data/alignment) should be retrieved using two pairs of brackets, as shown below.
alignmentLength(obj = obj[[1]], alignment_name = basename(obj_names[1]), Len =
100)
#Note: the following function unlink will delete the folders along with the contained
  files. Do not run this function except you want.
unlink(x = c("long_alignments", "short_alignments"), recursive = TRUE)
#run multiple objects in batch.
mapply(FUN = alignmentLength, obj, alignment_name = basename(obj_names), Len
  = 100)
#Note: the following function unlink will delete the folders along with the contained
  files. Do not run this function except you want.
unlink(x = c("long_alignments", "short_alignments"), recursive = TRUE)
#If run in parallel using multiple threads (e.g. two threads), set as below:
library(parallel)
cl = makeCluster(2)
clusterMap(cl, fun = alignmentLength, obj, alignment_name = obj_names, Len = 100)
stopCluster(cl)
#Note: the following function unlink will delete the folders along with the contained
  files. Do not run this function except you want.
unlink(x = c("long_alignments", "short_alignments"), recursive = TRUE)
```

anyShortseq: delete short sequences in alignment

Description

Sequences in alignment shorter than any of the two thresholds specified by either parameter *Len* or *LenP* are deleted.

Usage

```
anyShortseq(obj, alignment_name, Len = 50, LenP = 0.1, none_shortseq_data = F)
```

Arguments

obj	A variable of DNA alignment in FASTA format read in R.
alignment_name	The name of the DNA alignment (obj).
Len	A number specifying cutoff below which sequences with lengths are to be deleted from alignment.
LenP	A number in proportion from 0 to 1, also specifying cutoff below which sequences with lengths are to be deleted.
none_shortseq_data	Logical values TRUE or FALSE. If TRUE, output the alignments containing none short sequences to the designated folder too.

Details

Alignment in FASTA format can contain the four nucleotide symbols "ATGC" in both lower and upper cases. Degenerate (ambiguous) nucleotide symbols, e.g. “N”, are not counted in length calculation.

Value

Two objects including a reduced alignment with short sequences removed, and the document file recording the information about which alignment and short sequence were treated. If none_shortseq_data = TRUE, the intact alignments (without any short sequences that need deletion) will also be copied to a new folder.

Examples

```
library(magrittr)
library(utils)
obj_names <- dir(system.file("extdata", package = "alignmentFilter"), full.names =
  TRUE) %>% grep(pattern = "fasta$", value = TRUE)
obj = lapply(obj_names, readLines)
# But if your data is saved such as in a folder named mydata under disc “D:”, you
  can set the R working directory to the folder and read your data like this if the
  names of the data files ending with suffix “.fasta”.
setwd("D:/mydata")
```

```
obj_names <- list.files(pattern = ".fasta$")
obj <- lapply(obj_names, readLines)
# Run single object. Pay attention to that the data type of obj is list, so single object
  (alignment) should be retrieved using two pairs of brackets, as shown below.
anyShortseq(obj = obj[[1]], alignment_name = basename(obj_names[1]), Len =
  100)
#Note: the following function unlink will delete the folders along with the
  contained files. Do not run this function except you want.
unlink(x = c("short_seq_information", "rm_shortseq_data", "none_shortseq_data"),
  recursive = TRUE)
#run multiple objects in batch.
mapply(FUN = anyShortseq, obj, alignment_name = basename(obj_names), Len =
  100)
#Note: the following function unlink will delete the folders along with the
  contained files. Do not run this function except you want.
unlink(x = c("short_seq_information", "rm_shortseq_data", "none_shortseq_data"),
  recursive = TRUE)
#If run in parallel using multiple threads (e.g. two threads), set as below:
library(parallel)
cl = makeCluster(2)
clusterMap(cl, fun = anyShortseq, obj, alignment_name = basename(obj_names),
  Len = 100)
stopCluster(cl)
#Note: the following function unlink will delete the folders along with the
  contained files. Do not run this function except you want.
unlink(x = c("short_seq_information", "rm_shortseq_data", "none_shortseq_data"),
  recursive = TRUE)
```

degap: remove gappy sites (columns)

Description

The gappy sites (columns) in alignment having gaps equal or over than a given threshold are removed.

Usage

```
degap(obj, alignment_name, datatype = "DNA", p = 0.5, p_flankgap = 0.5,
  check_flank_gaps = T, none_gappy_data = F)
```

Arguments

obj	A variable of DNA alignment in FASTA format.
alignment_name	The name of the DNA alignment (obj).
datatype	A character string, either "DNA" or "codon".

p	A number in proportion from 0 to 1, specifying the cutoff value, equal or over which sites (or codons) having gap ratio are to be removed.
p_flankgap	A number in proportion from 0 to 1, specifying the cutoff value, equal or over which the terminal gappy sites (or codons) at both ends having gap ratio are to be removed. The terminal gappy sites or codons at both ends are defined as continuous gappy sites that start or end from the first or last site in any one sequence in alignment.
check_flank_gaps	Logical values (TRUE or FALSE) to decide whether check and treat the terminal gaps independently.
none_gappy_data	Logic values TRUE or FALSE. If TRUE, output the data of low or none gappy sites that need not remove to the designated folder. Default is FALSE.

Details

If the datatype is codon and if any site (column) in a codon has a proportion of gaps equal or over the specified threshold, the codon as a whole would be removed. The function would check if alignment is of a length of multiple of three but not check premature stop codon if the datatype selected as "codon". If parameters *p_flankgap* or *p* set to zero and there is not any gap, no site (column) would be deleted. If there is any sequence completely composed of gaps, it would be assumed to contain none flank gaps as unable to be regularly matched in identifying flank gaps using our regular expression at this situation. The program checks and removes the flank (terminal) gaps first if *check_flank_gaps* is set to TRUE, and after that all other gaps are checked according to the cutoff setting to the parameter *p*.

Value

degapped DNA alignments.

Examples

```
library(magrittr)
obj_names <- dir(system.file("extdata",package = "alignmentFilter"), full.names =
  TRUE) %>% grep(pattern = "fasta$", value = TRUE)
obj <- lapply(X = obj_names, FUN = readLines)
# But if your data is saved such as in a folder named mydata under disc “D:”, you can
  set the R working directory to the folder and read your data like this if the names of
  the data files ending with suffix “.fasta”.
setwd("D:/mydata")
obj_names <- list.files(pattern = ".fasta$")
```

```
obj <- lapply(obj_names, readLines)
#Run single object. Pay attention to that the data type of obj is list, so single object
   (alignment) should be retrieved using two pairs of brackets, as shown below.
degap(obj = obj[[1]], alignment_name = basename(obj_names[1]), p = 0.6)
#Note: the following function unlink will delete the folders along with the contained
   files. Do not run this function except you want.
unlink(x = c("degaps", "none_gappy_data"), recursive = TRUE)
#Run multiple objects in batch
mapply(FUN = degap, obj, alignment_name = basename(obj_names), p = 0.6)
#Note: the following function unlink will delete the folders along with the contained
   files. Do not run this function except you want.
unlink(x = c("degaps", "none_gappy_data"), recursive = TRUE)
#If run in parallel using multiple threads (e.g. two threads), set as below:
library(parallel)
cl = makeCluster(2)
clusterMap(cl, fun = degap, obj, alignment_name = basename(obj_names), p = 0.6)
stopCluster(cl)
#Note: the following function unlink will delete the folders along with the contained
   files. Do not run this function except you want.
unlink(x = c("degaps", "none_gappy_data"), recursive = TRUE)
```

***maskSegment*: mask overly-divergent/ambiguously-aligned segments in alignment

Description

To identify and mask overly-divergent segments in alignment is based on raw similarity scores and sliding windows, with group(s) of segments (sequences) in which all has low similarities, i.e. statistically falling in random similarity, to any segment of the main group masked with "N". Note: Only the definite nucleotide symbols, i.e. “ATGC”, will be replaced with “N” (default) and degenerate bases and gaps will be left unchanged.

Usage

```
maskSegment(obj, alignment_name, type = "DNA", prob = 0.01, window_width =
   15, window_step = 3, replacement = "N", mask_shortstrings = T,
   maxLen_shortstrings = 3, checkcase = F, copy_unfiltered = F)
```

Arguments

obj	A variable of DNA alignment read in R.
alignment_name	The name of the DNA alignment.
type	A character string either "DNA" or "codon".
prob	A decimal among a recommended set of values including 0.05,

	0.01, 0.001, and 0.0001, which controls the stringency. Bigger value means less stringency, probably filtering less segments than that with smaller values. Other values beyond the recommended are dissuaded. Note: The theoretically possible intervals of the parameter is between 0 and 1 (But if the extremely small value is set, for example zero, every segments would be masked. Conversely, if the possibly maximum value, i.e. 1, is set, nearly nothing except segments with none identical site would be filtered.
window_width	A number controlling the width of sliding windows. Larger value could speed up but may fail to identify short overly-divergent segment. Default is strongly recommended.
window_step	A number controlling the step size of sliding windows. Larger value could speed up but may decrease power of identifying (short) overly-divergent segments. Note this value should at least be lower than the window width, or some sites (columns) would be skipped, i.e. not included in any sliding window for evaluation. Default is strongly recommended.
replacement	A state (symbol) usually comprising "N" (default), "-" (gap) or "?" which the ambiguously-aligned residues would be overwritten with.
mask_shortstrings	Logical values TRUE or FALSE to set whether short nucleotide strings (contiguous nucleotide strings separated by gap) need to be masked (i.e. replaced by symbols defined by the parameter replacement) or not before conducting formal evaluation of ambiguously-aligned residues using grouping-regrouping algorithm across sliding windows.If mask_short_strings = T (default), the short strings (with length equal or shorter than the given value of the parameter *maxLen_shortstrings*) would be initially masked despite how similar (identical) they are aligned to the other sequences.
maxLen_shortstrings	Numeric value equal and under which the short nucleotide strings would be initially masked. This make effect only under *mask_shortstrings* = T. The default is three contiguous nucleotide strings, and larger value is strongly dissuaded in case of correctly-aligned strings are directly masked. Longer ambiguously-aligned strings could be identified by the following grouping-regrouping algorithm.
checkcase	A logical value TRUE or FALSE. If TRUE, the nucleotides in lower case are rewritten in upper case.
copy_unfiltered	Logic values TRUE or FALSE. If TRUE, a folder is set up to accommodate the unfiltered alignments.

Details

The default settings, particularly with higher stringency (lower value of the parameter *prob*) can work well enough even for divergent sequences from across higher taxonomic units, e.g. phylogenetic reconstruction of green plants, if the taxa are densely sampled. But if higher taxonomic units with sparse sampling, relaxed values (e.g. prob = 0.05) should better be adopted. Otherwise, the divergent but correct alignments may be masked, leaving few informative sites. The information about which alignments and taxa and which segments (sliding windows) are masked are recorded in the output information files. Note if DNA type is codon (type = codon), the window width (window_width) and step size (window_step) must be set with numbers that are multiple of 3.

Value

Two sorts of objects including (masked or unfiltered) alignments and files about information of position(s) of the masked segments in alignment and the corresponding taxa names.

Examples

```
library(magrittr)
library(utils)
obj_names = dir(system.file("extdata", package = "alignmentFilter"), full.names =
  TRUE) %>% grep(pattern = "fasta$", value = TRUE)
obj = lapply(X = obj_names, FUN = readLines)
# But if your data is saved such as in a folder named mydata under disc “D:”, you can
  set the R working directory to the folder and read your data like this if the names of
  data files ending with suffix “.fasta”.
setwd("D:/mydata")
obj_names <- list.files(pattern = "fasta$")
obj <- lapply(obj_names, readLines)
# Run single object. Pay attention to that the data type of obj is list, so single object
  (alignment) should be retrieved using two pairs of brackets, as shown below.
maskSegment(obj = obj[[1]], alignment_name = obj_names[1], prob = 0.05)
#Note: the following function unlink will delete the folders along with the contained
  files. Do not run this function except you want.
unlink(x = c("mask_alignment", "mask_information", "unfiltered_data"), recursive =
  TRUE)
#Run multiple objects in batch.
mapply(FUN = maskSegment, obj = obj, alignment_name = obj_names, prob = 0.05)
#Note: the following function unlink will delete the folders along with the contained
  files. Do not run this function except you want.
unlink(x = c("mask_alignment", "mask_information", "unfiltered_data"), recursive =
  TRUE)
```

```
#If run in parallel using multiple threads (e.g. two threads), set as below:
library(parallel)
cl = makeCluster(2)
clusterMap(cl, fun = maskSegment, obj, alignment_name = obj_names, prob = 0.05)
stopCluster(cl)
#Note: the following function unlink will delete the folders along with the contained
   files. Do not run this function except you want.
unlink(x = c("mask_alignment", "mask_information", "unfiltered_data"), recursive =
   TRUE)
```

revComplement: identify and adjust reverse complement sequences in alignment

Description

The common (phylogenetic) pipelines may bring reverse complementary sequences in alignment. This function is to identify and adjust possible fewer sequences that are homologous but in reverse direction (minus) to other majority of sequences in forward direction (plus).

Usage

```
revComplement(obj_blast, obj, filename, none_revcomplement_data = F)
```

Arguments

obj_blast	A variable of the blastout file read in R.
obj	A variable of a set of DNA sequences, either aligned or not, read in R.
filename	The name of the input DNA sequence file.
none_revcomplement_data	Logical value TRUE or FALSE. If TRUE, output the data originally containing none reverse complementary sequences to the designated folder too.

Details

Before running this function, users need to prepare two sorts of input files. Besides sequence files (obj), there need output files (obj_blast) that are generated by running the program BLAST++ with sequence files as the input. The brief procedure of running BLAST++ is as followings: 1) withdraw one sequence from each of sets of homologous sequences as the query; 2) make corresponding databases of individual datasets using BLAST++; 3) running BLAST++ to produce the output files. After that, running the function "revComplement*"* with the blast output files along with sequence datasets to decide whether there exists any homologous reverse complementary sequences and reverse complement them if

there is. If there is any homologous reverse complementary sequences contained, the sequences in one direction and less numbers are to be reversely complemented to fit with the majority sequences in the opposite direction, regardless of whether they are really plus or minus strand when they are protein coding genes. Ordinarily users should run this function before or after alignment, but it is also OK to not run this function unless some sequences are almost entirely masked by using the function “maskSegment”, suggesting homologous reverse complementary sequences are probably included. Note: if special symbols other than “ATGC” such as "!" contained, they will be deleted directly from the output sequences.

Value

Alignments with reverse complementary sequences adjusted.

Examples

```
library(magrittr)
library(Biostrings)
blastoutFilename <- system.file("extdata", package = "alignmentFilter") %>%
  dir(pattern = "results_OG", full.names = TRUE)
obj_blast <- lapply(X = blastoutFilename, readLines)
filename <- system.file("extdata" ,package = "alignmentFilter") %>%
  dir(pattern = "_revcom.fasta$", full.names = TRUE)
obj <- lapply(X = filename, readDNAStringSet)
# But if your data is saved such as in a folder named mydata under disc “D:”, you can set the R working directory to the folder and read your data like this if the blast out files begin with prefix results_OG and sequence data file names ending with suffix “revcom.fasta”.
setwd("D:/mydata")
blastoutFilename <- list.files(pattern = "^results_OG")
obj_blast <- lapply(X = blastoutFilename, readLines)
filename <- list.files(pattern = "revcom.fasta$")
obj <- lapply(X = filename, readDNAStringSet)
#Run single object. Pay attention to that the data type of obj is list, so single object/alignment should be retrieved using two pairs of brackets, as shown below.
revComplement(obj_blast = obj_blast[[1]], obj[[1]], filename = filename[1])
#Note: the following function unlink will delete the folders along with the contained files. Do not run this function except you want.
unlink(x = c("reverse_complement_data", "none_revcomplement_data"), recursive = TRUE)
#Run multiple objects in batch.
mapply(FUN = revComplement, obj_blast = obj_blast, obj = obj, filename = filename)
#Note: the following function unlink will delete the folders along with the contained files. Do not run this function except you want.
```

```
unlink(x = c("reverse_complement_data", "none_revcomplement_data"), recursive =
  TRUE)
#If run in parallel using multiple threads (e.g. two threads), set as below:
library(parallel)
cl = makeCluster(2)
clusterMap(cl, fun = revComplement, obj_blast = obj_blast, obj = obj, filename =
  filename)
stopCluster(cl)
#Note: the following function unlink will delete the folders along with the contained
  files. Do not run this function except you want.
unlink(x = c("reverse_complement_data", "none_revcomplement_data"), recursive =
  TRUE)
```

Appendices

Order of running the five functions of "alignmentFilter":

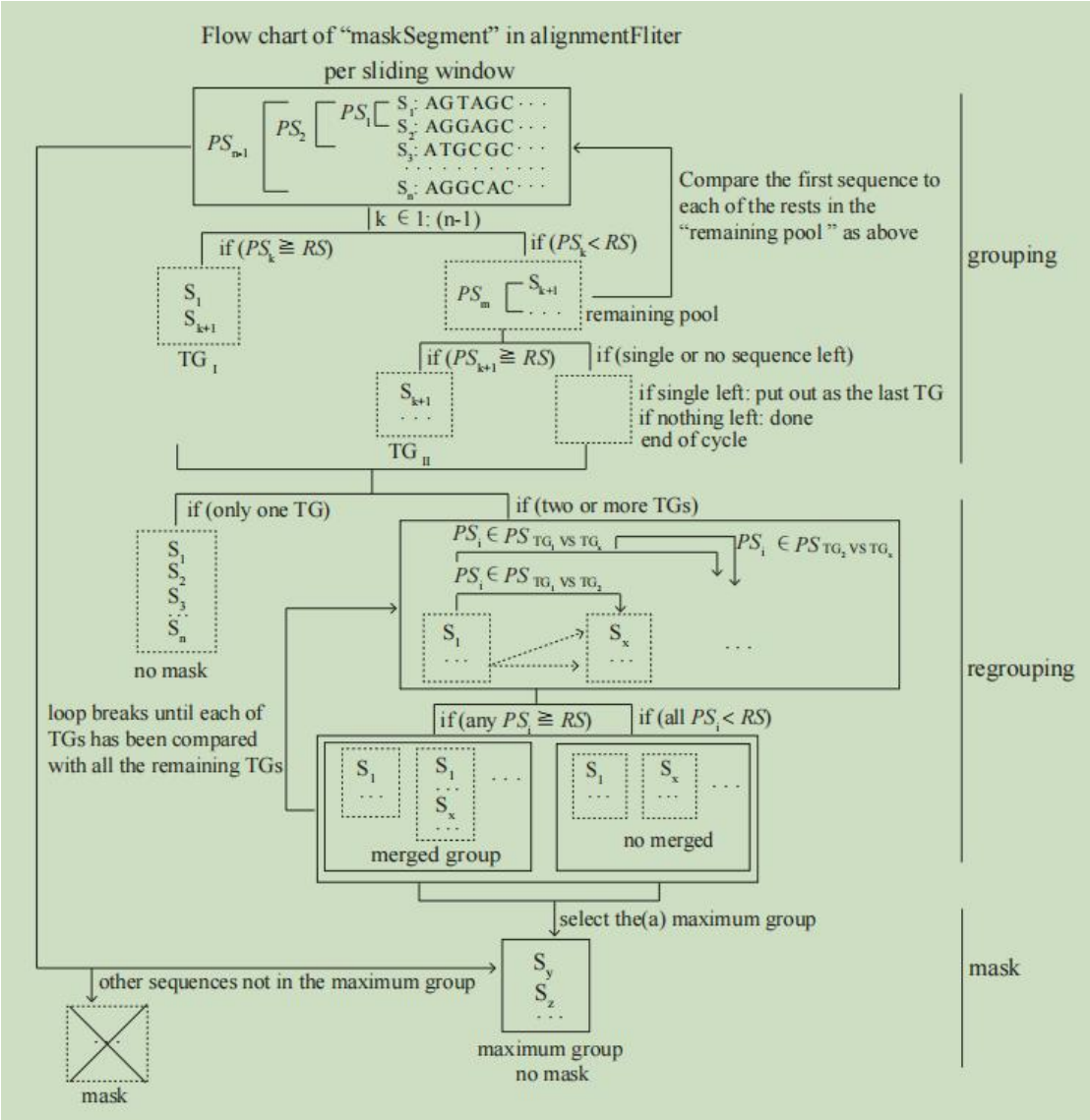
They can be coordinately used in an recommended order of "revComplement" (adjusting possible homologous sequences in the opposite direction before or after aligning, which is better to be followed with realigning if any reverse complementary sequence is removed), "alignmentLength*"* (excluding short alignments), "anyShortseq*"* (removing short sequences in alignment, which is also better to be followed with a realigning process), "degap*"* (excluding gappy sites), and "maskSegment*"* (masking ambiguously-aligned segments). After using "maskSegment", the functions "anyShortSeq" and "degap" can be considered to be used again if necessary in case of some sequences or columns are mostly masked and need to be removed. But some of them can also be selectively used according to factual requirement or they can be run in alternative orders. For example, if prior knowledge (or observations from results running the core function "maskSegment") tells none short alignments, short sequences or homologous sequences in both directions, the related functions can be skipped.

Rationale of "maskSegment*"* in alignmentFilter:

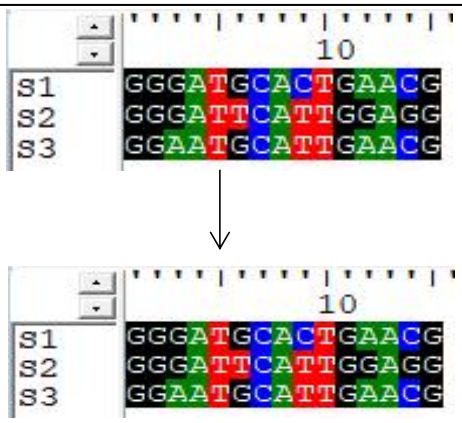
The existing alignment quality filtering methods can be grossly divided into two categories, i.e. divergence-based and stability/consistency-based, according to their fundamental rationales. The core function "maskSegment*"* of alignmentFilter is divergence-based method, which identify and mask overly-divergent segments statistically falling in random similarity comparing to all the sequences from the inferred major group of sequences. The probability (*P*) of similarity of two random nucleotide sequences in length (*n*) can be formulated as $P = \sum_{k=0}^{0 \le k \le n} \binom{n}{k} (3/4)^k (1/4)^{n-k}$, in which, *k* denotes the number of sites (columns) that are different in both sequences. According to the formular, a hidden parameter in "maskSegment", i.e. random similarity cutoff (*RS*), can be further calculated under

different given probabilities: $RS = (n-k)/n =$

$\begin{cases} 0.80, k = 3, n = 15, P < 0.0001 \\ 0.67, k = 5, n = 15, P < 0.001 \\ 0.60, k = 6, n = 15, P < 0.01 \\ 0.53, k = 7, n = 15, P < 0.05 \end{cases}$. For example, if the stringency parameter *Prob* was set to 0.01 (default), a paired sequences in alignment would be divided into two groups when they have raw similarity value (a calculated hidden variable, $PS = (n-k)/n = \frac{\sum_{\substack{i \in 1:n \\ x_i = y_i}} 1}{\sum_{\substack{i \in 1:n \\ x_i = y_i | x_i != y_i}} 1}$, *x*, *y* respectively represent each sequence of the sequence pair with length *n* in alignment and the iteration variable *i* denotes *i*th site in the sequence) lower than 0.60, they would be clustered into a same group otherwise. The newly devised grouping-regrouping algorithm implemented in "maskSegment" has advantages of high computational efficiency and always reaching the optimal solution under the implemented criterion/rationale. A flowchart of "maskSegment" is as following:



Here below are simplified cases (A/B/C/D) to show how grouping-regrouping algorithm of "maskSegment" works:



one group, no mask

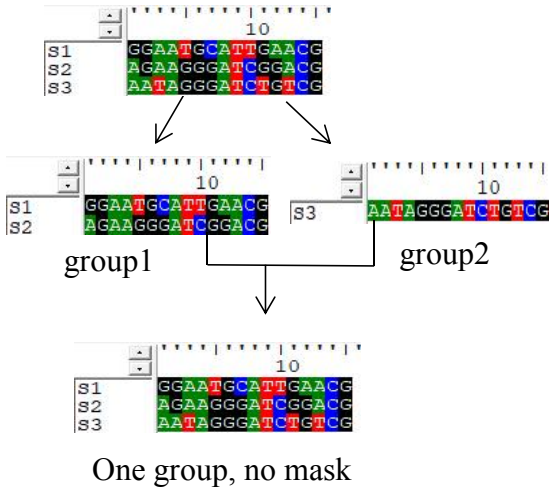
if *prob* = 0.001 (*RS* = 0.67, *n* = 15)

grouping:

PS_{12}=11/15≈0.73>*RS*

PS_{13}=13/15≈0.87>*RS*

A: Grouping yields only one group, need not to do regrouping and masking. Computation complexity is O(NL).



One group, no mask

if *prob = 0.01* (*RS*=0.6, *n* =15)

grouping:

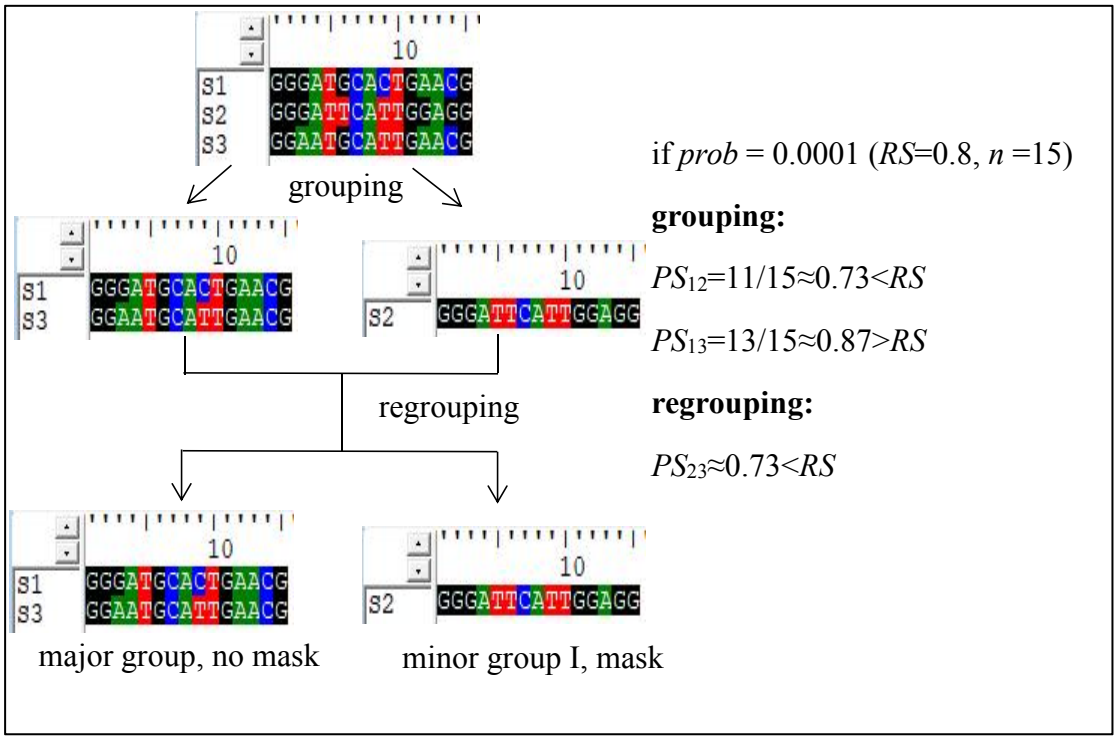
PS_{12}=10/15≈0.67>*RS*

PS_{13}=8/15≈0.53<*RS*

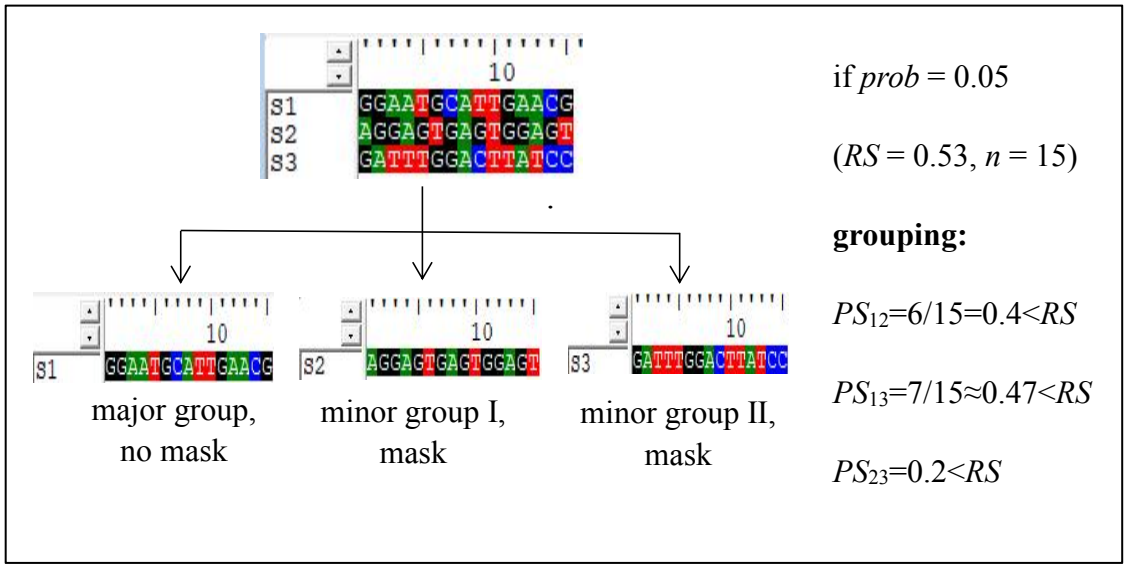
regrouping:

PS_{23}≈ 0.7>*RS*

B: The grouping process yields two groups, but regrouping clusters them into one group.
. Computation complexity is O(NL)~O(N^2L), which would conventionally approach O(NL). O(N^2L) is just a theoretical upper boundary that is very unlikely in realistic cases.



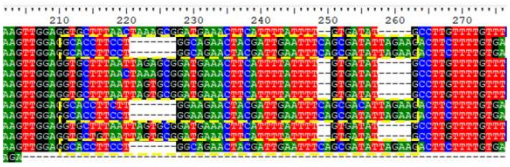
C: Both the grouping and regrouping processes yield two groups. The major group would not be masked while the minor group(s) would be masked. Computation complexity is O(NL)~O(N^2L), which would conventionally approach O(NL). O(N^2L) is just a theoretical upper boundary that is very unlikely in realistic cases.



D: The grouping process suggests each of individual sequences should be placed in an independent group, that is, not any two sequences are divided into a same group. As every sequence in every group has been compared with each other, so that there needs not regrouping in this simplified case. The (one of) major group (the first group in this specific simplified case as they each have one sequence) would not be masked while the minor groups (the latter two) would be masked. At this very specific case, Computation complexity reaches the theoretical maximum O(N^2L), exactly 1/2LN(N-1).

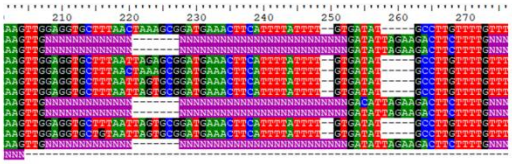
Comparisons of “maskSegment”/alignmentFilter with the other analogues, visualizing the effects of the alignment-filtering methods:

1) In context of very high frequent alignment errors (fraction of the falsely-aligned sequences to the slightly more correctly-aligned sequences in a sliding window).

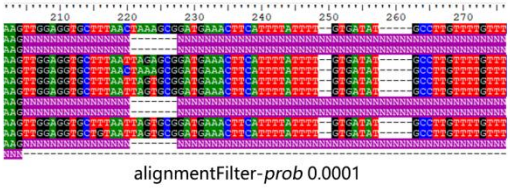


Unfiltered

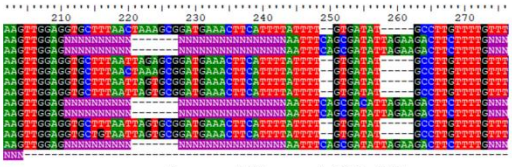
(The ambiguously-aligned segments are enclosed by yellow box)



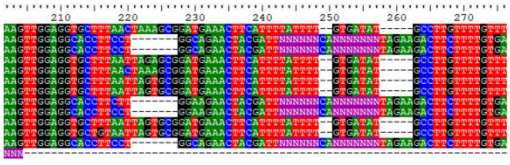
alignmentFilter-*prob* 0.001



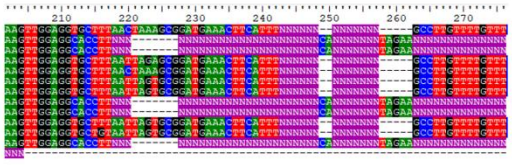
alignmentFilter-*prob* 0.0001



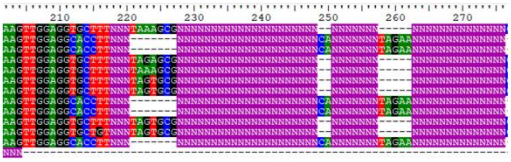
alignmentFilter-*prob* 0.01, 0.05



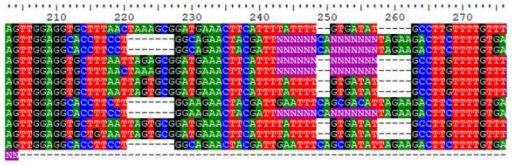
Hot-cutoff0.5, 0.6



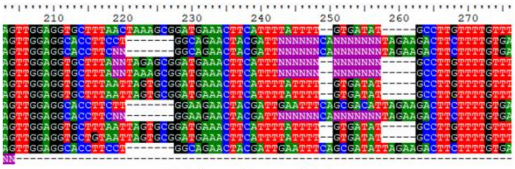
Hot-cutoff0.7



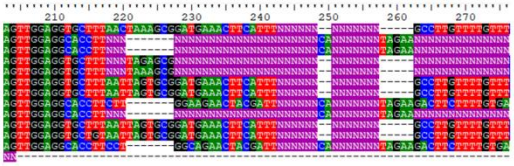
Hot-cutoff0.8



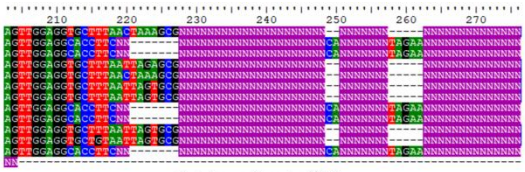
Guidance2-cutoff0.5



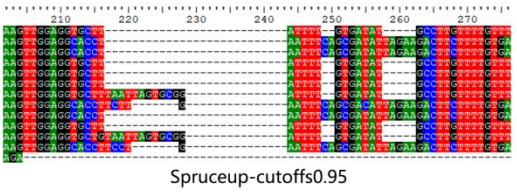
Guidance2-cutoff0.6



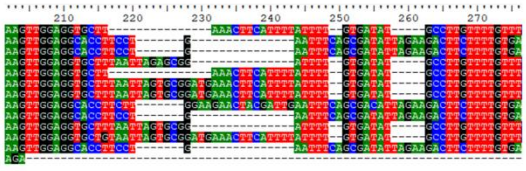
Guidance2-cutoff0.7



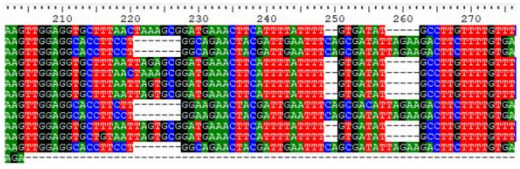
Guidance2-cutoff0.8



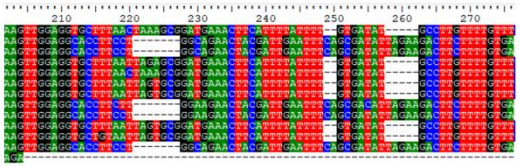
Spruceup-cutoffs0.95



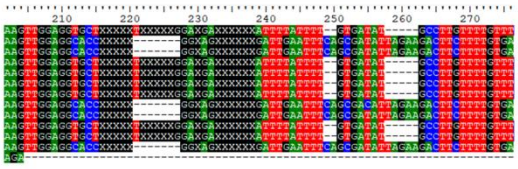
Spruceup-cutoffs0.97



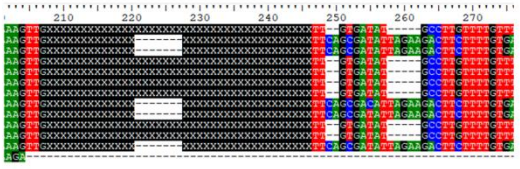
Spruceup-cutoffs0.99



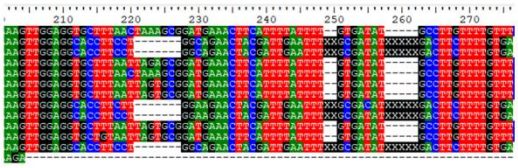
TAPER, HmmCleaner



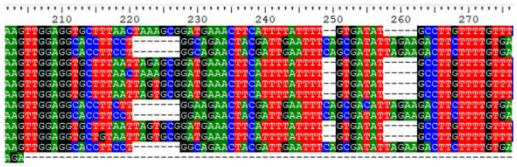
trimAl-*st* 0.7



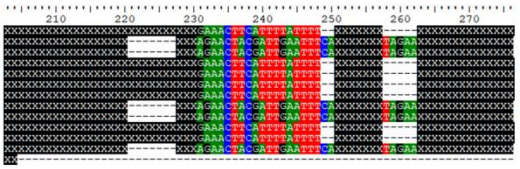
trimAl-*st* 0.8



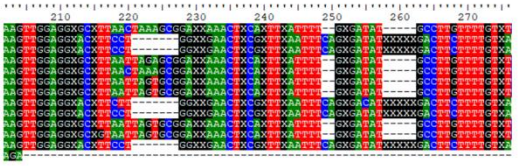
trimAl-*st* automated1



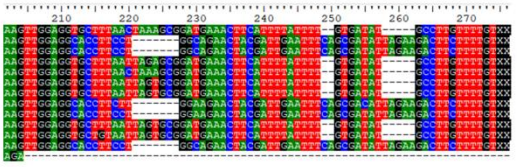
trimAl-*st* 0.5, 0.6



Fasparser2

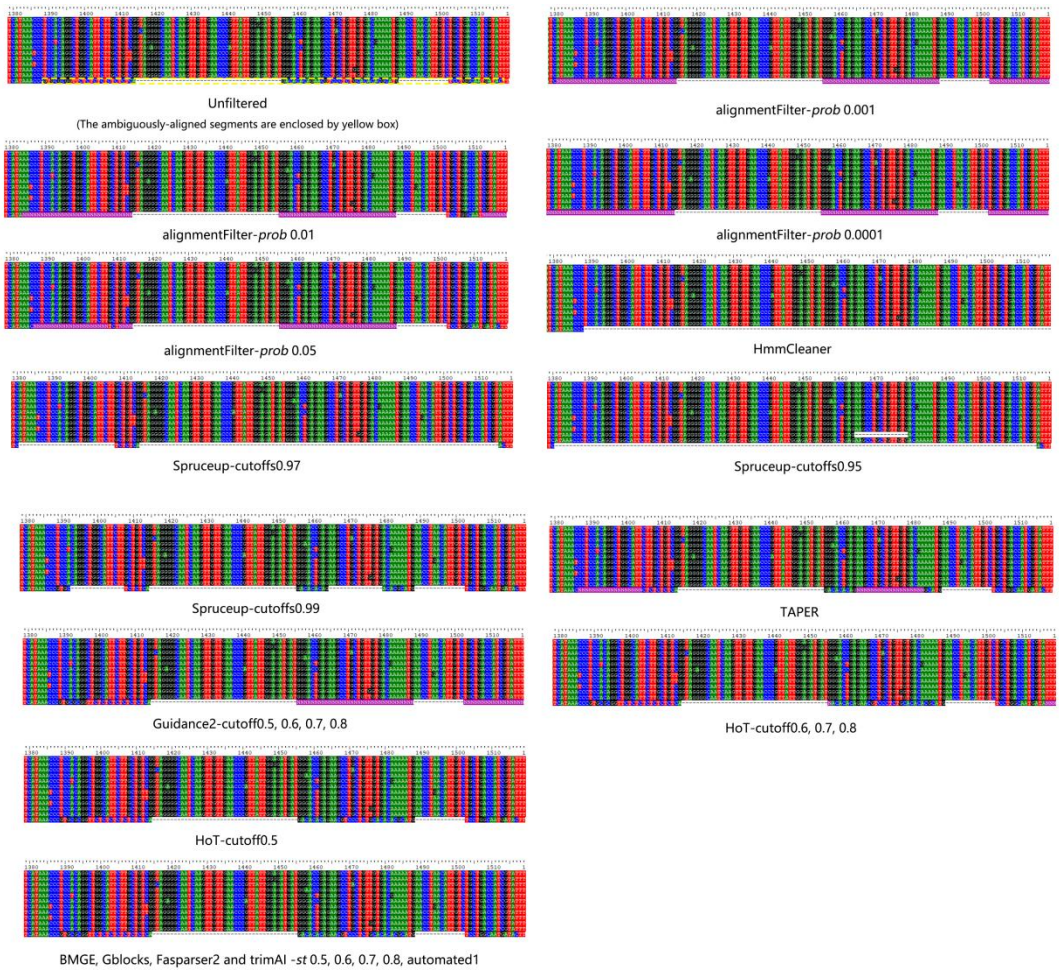


BMGE

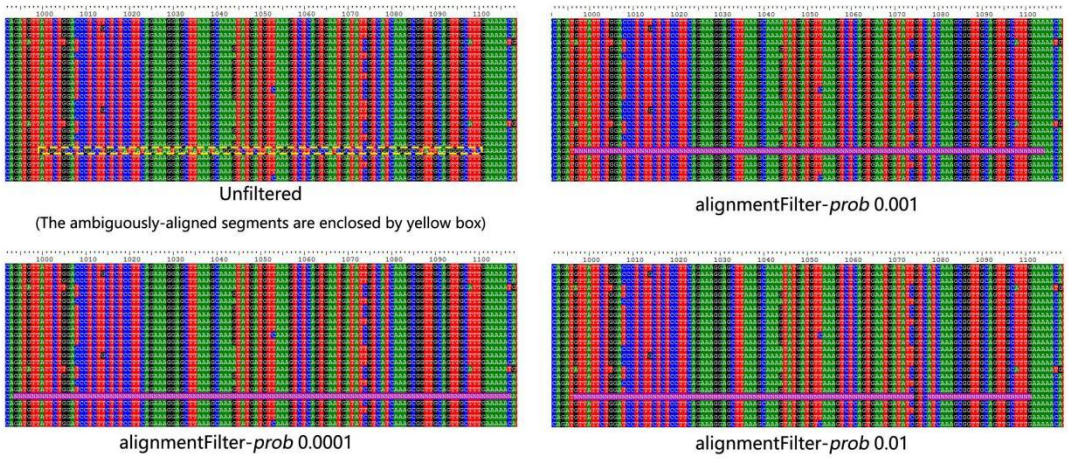


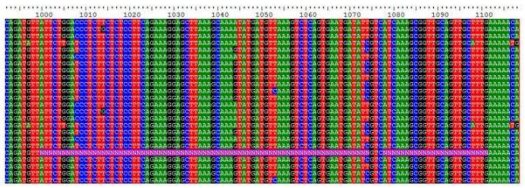
Gblocks

2) In context of very short alignment errors.

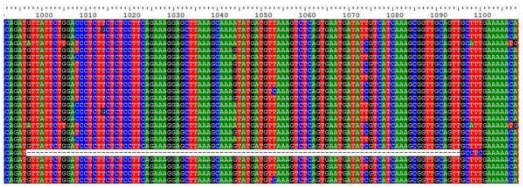


3) In context of very low frequent alignment errors.

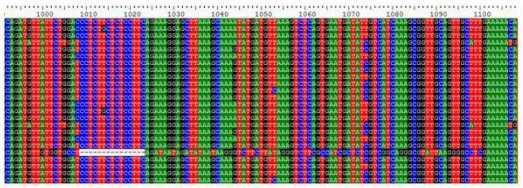




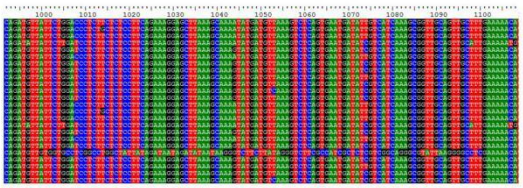
alignmentFilter-*prob* 0.05



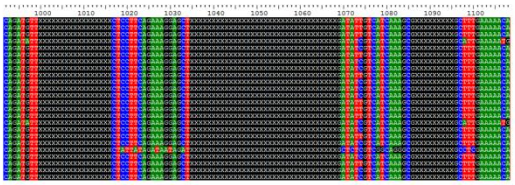
Spruceup-cutoffs0.95



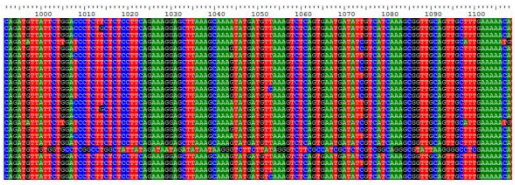
Spruceup-cutoffs0.97



Spruceup-cutoffs0.99

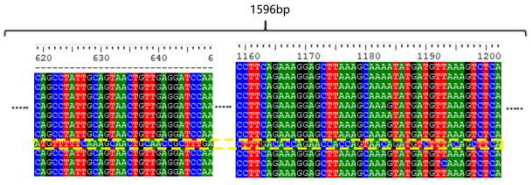


Fasparser2

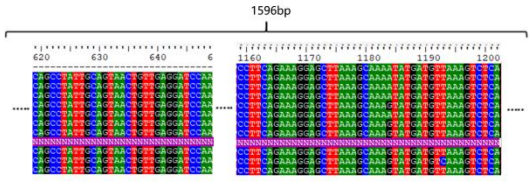


BMGE, Gblocks and trimAl-st0.5, 0.6, 0.7, 0.8, automated

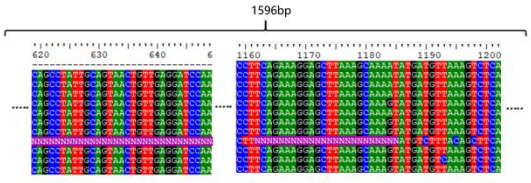
4) In context of very long alignment errors.



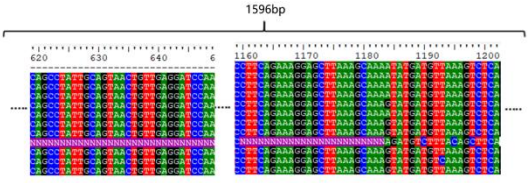
Unfiltered
(The ambiguously-aligned segments are enclosed by yellow box)



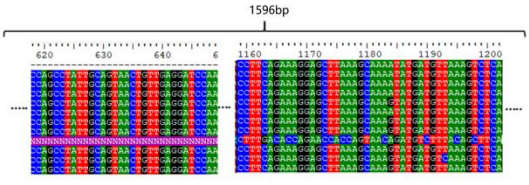
alignmentFilter-*prob* 0.01, 0.001, 0.0001



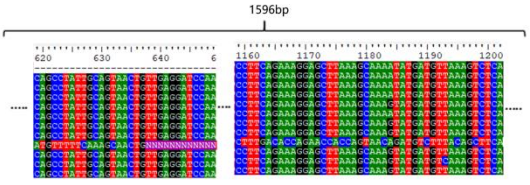
alignmentFilter-*prob* 0.05



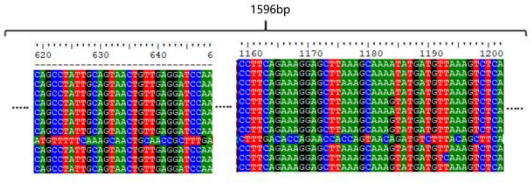
TAPER



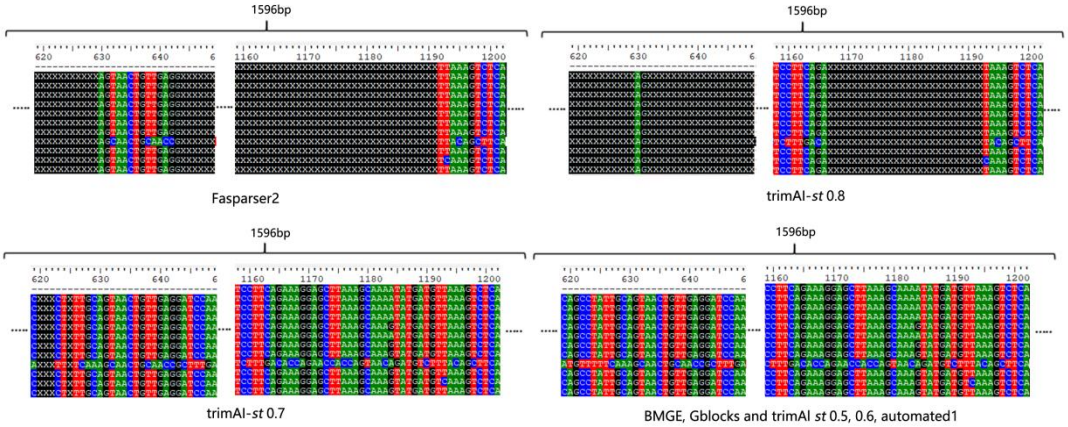
Guidance2-cutoff0.6, 0.7, 0.8



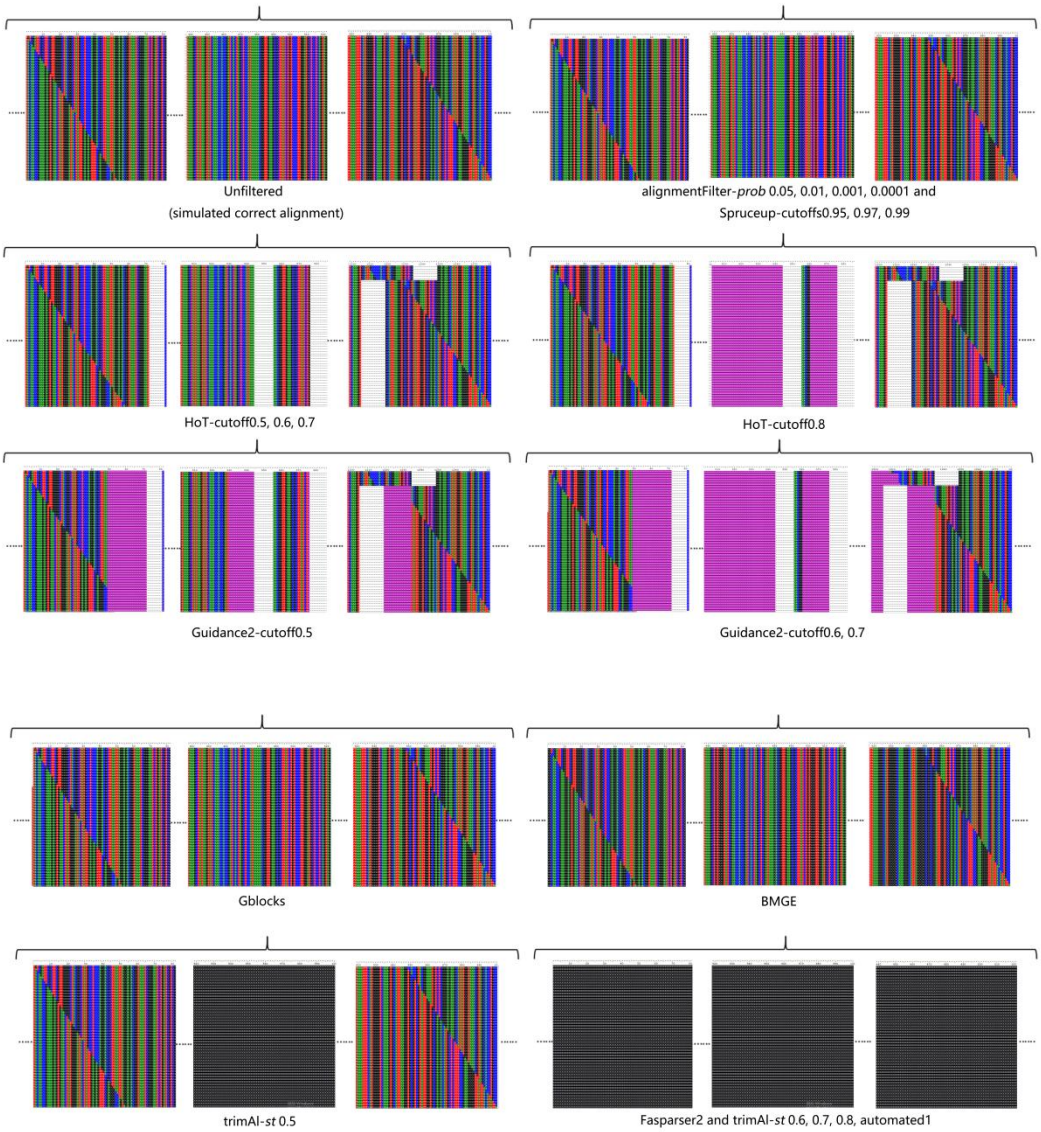
Guidance2-cutoff0.5

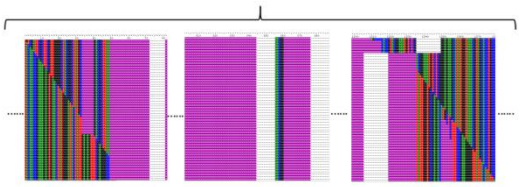


HoT-cutoff0.5, 0.6, 0.7, 0.8, HmmCleaner and Spruceup-cutoffs0.95, 0.97, 0.99

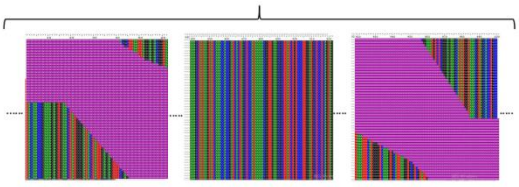


5) In context of completely correctly-aligned alignment in which any sequence (except the first one) is derived from one mutation at a successive site from the preceding sequence. None residues should be masked.

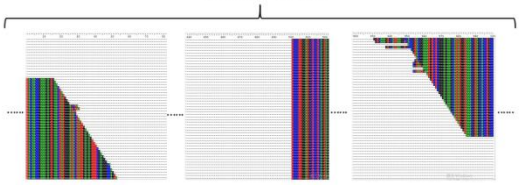




Guidance2-cutoff0.8



TAPER



Hmmcleaner