Package 'DNABarcodes'

November 11, 2026

Type Package

Title A tool for creating and analysing DNA barcodes used in Next Generation Sequencing multiplexing experiments

Version 1.42.0

Date 2014-07-23

Author Tilo Buschmann <tilo.buschmann.ac@gmail.com>

Maintainer Tilo Buschmann <tilo.buschmann.ac@gmail.com>

Description The package offers a function to create DNA barcode sets capable of correcting insertion, deletion, and substitution errors. Existing barcodes can be analysed regarding their minimal, maximal and average distances between barcodes. Finally, reads that start with a (possibly mutated) barcode can be demultiplexed, i.e., assigned to their original reference barcode.

License GPL-2

Imports Rcpp (>= 0.11.2), BH

LinkingTo Rcpp, BH

Depends Matrix, parallel

Suggests knitr, BiocStyle, rmarkdown

VignetteBuilder knitr

biocViews Preprocessing, Sequencing

NeedsCompilation yes

git_url https://git.bioconductor.org/packages/DNABarcodes

git_branch RELEASE_3_23

git_last_commit e6484ea

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Contents

DNABarcodes-package . . . 2
analyse.barcodes . . . 3
barcode.set.distances . . . 5
create.dnabarcodes . . . 6
create.pool . . . 10
demultiplex . . . 11
distance . . . 12
mutatedReads . . . 13
supplierSet . . . 14

Index **15**

`DNABarcodes-package` *Create and analyse DNA barcode sets that are capable of error correction.*

Description

The package offers a function to create DNA barcode sets capable of correcting substitution errors or insertion, deletion, and substitution errors. Existing barcodes can be analysed regarding their minimal, maximal and average distances between barcodes. Finally, reads that start with a (possibly mutated) barcode can be demultiplexed, i.e. assigned to their original reference barcode.

Details

Package:	DNABarcodes
Type:	Package
Version:	0.1
Date:	2014-07-23
License:	GPL-2

The function `create.dnabarcodes` creates a set of barcodes of equal length that satisfies some wished criteria regarding error correction.

After sequencing the DNA/RNA material, the researcher will have a set of reads that start with a (possibly mutated) barcode. For Illumina HiSeq, this is the index read. For PacBio, this is the read itself (with some other complications). The function `demultiplex` can then be used to assign reads to their original reference barcodes. `demultiplex` will correct mutations in a best-effort way.

Existing sets of barcodes (e.g. supplied by a manufacturer) can be analysed with functions `analyse.barcodes` and `barcode.set.distances`.

The advantage of this package over using already available barcode sets in the scientific community is the ability to flexibly generate new barcode sets of different properties. For example, `create.dnabarcodes` can use a pre-existing barcode library as a candidate set for a better barcode set. In another example, a higher distance (e.g., `dist = 4`) can be used. Such a parameter setting would possibly increase the error detection property of the code as well as the average barcode distance, increasing the probability of guessing a barcode during demultiplexing.

Author(s)

Tilo Buschmann (tilo.buschmann.ac@gmail.com)

References

Buschmann, T. and Bystrykh, L. V. (2013) Levenshtein error-correcting barcodes for multiplexed DNA sequencing. BMC bioinformatics, 14(1), 272. Available from http://www.biomedcentral.com/1471-2105/14/272.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. In Soviet physics doklady (Vol. 10, p. 707).

Hamming, R. W. (1950). Error detecting and error correcting codes. Bell System technical journal, 29(2), 147-160.

Conway, J. and Sloane, N. (1986) Lexicographic codes: error-correcting codes from game theory. Information Theory, IEEE Transactions on, 32(3), 337-348.

Pattabiraman, B., Patwary, M. M. A., Gebremedhin, A. H., Liao, W. K. and Choudhary, A. (2013) Fast algorithms for the maximum clique problem on massive sparse graphs. In Algorithms and Models for the Web Graph (pp. 156-169). Springer International Publishing.

Ashlock, D., Guo, L. and Qiu, F. (2002) Greedy closure evolutionary algorithms. In Computational Intelligence, Proceedings of the World on Congress on (Vol. 2, pp. 1296-1301). IEEE.

Brouwer, A. E., Shearer, L. B. and Sloane, N. I. A. (1990) A new table of constant weight codes. In IEEE Trans Inform Theory.

Examples

```
# Create Sequence Levenshtein Barcodes with the default heuristic
dnabarcodes1 <- create.dnabarcodes(5, metric="seqlev")

# Create Sequence Levenshtein Barcodes with a better, but slower heuristic
dnabarcodes2 <- create.dnabarcodes(5, metric="seqlev", heuristic="ashlock")
```

---

analyse.barcodes *Analyse Sets of Barcode*

Description

The function analyses the properties of sets of (potential) barcodes. For various metrics, minimal, maximal and average distances are calculated and hints on error correction capabilities of the code are given.

Usage

```
analyse.barcodes(barcodes, metric = c("hamming", "seqlev", "levenshtein"), cores=detectCores()/2, cost
```

Arguments

barcodes	A vector of characters that represent the barcodes. All barcodes must be of equal length and consist only of letters A, C, G, and T. Lower case letters are allowed but do not make a difference.
metric	A vector of one or more metric names whose distances shall be calculated for the barcode set. Default is to use all of them.
cores	The number of cores (CPUs) that will be used for parallel (openMP) calculations.
cost_sub	The cost weight given to a substitution.
cost_indel	The cost weight given to insertions and deletions.

Value

A data frame of properties of the barcode set with the following meanings:

Columns: The first column contains a description of the barcode set properties. Each next column names the metric.

Rows: "Mean Distance", "Median Distance", "Minimum Distance", "Maximum Distance", "Guaranteed Error Correction", "Guaranteed Error Detection"

References

Buschmann, T. and Bystrykh, L. V. (2013) Levenshtein error-correcting barcodes for multiplexed DNA sequencing. BMC bioinformatics, 14(1), 272. Available from http://www.biomedcentral.com/1471-2105/14/272.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. In Soviet physics doklady (Vol. 10, p. 707).

Hamming, R. W. (1950). Error detecting and error correcting codes. Bell System technical journal, 29(2), 147-160.

See Also

barcode.set.distances

Examples

```
barcodes <- c("ACG", "CGT", "TGC")
analyse.barcodes(barcodes)
```

```
##
##                 Description  hamming   seqlev levenshtein
##               Mean Distance 2.666667 1.666667    2.333333
##             Median Distance 3.000000 2.000000    2.000000
##            Minimum Distance 2.000000 1.000000    2.000000
##            Maximum Distance 3.000000 2.000000    3.000000
## Guaranteed Error Correction 0.000000 0.000000    0.000000
##  Guaranteed Error Detection 1.000000 0.000000    1.000000
```

barcode.set.distances *Calculate distances between each barcode pair of a barcode set.*

Description

The function calculates the distance between each pair of a set of barcodes. The user may choose one of several distance metrics ("hamming", "seqlev", "levenshtein")).

Usage

```
barcode.set.distances(barcodes, metric=c("hamming", "seqlev", "levenshtein"), cores=detectCores()/2,
```

Arguments

barcodes	A set of barcodes (as vector of characters)
metric	The distance metric which should be calculated.
cores	The number of cores (CPUs) that will be used for parallel (openMP) calculations.
cost_sub	The cost weight given to a substitution.
cost_indel	The cost weight given to insertions and deletions.

Details

The primary purpose of this function is the analysis of barcode sets. Seeing the individual paired barcode distances helps to understand which pairings are exceptionally similar and which barcodes have a smaller or higher average distance to other barcodes.

Details if the distance metrics can be found in the man page of create.dnabarcodes.

Value

A symmetric Matrix of distances between each pair of barcodes with zeros on the main diagonal.

References

Buschmann, T. and Bystrykh, L. V. (2013) Levenshtein error-correcting barcodes for multiplexed DNA sequencing. BMC bioinformatics, 14(1), 272. Available from http://www.biomedcentral.com/1471-2105/14/272.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. In Soviet physics doklady (Vol. 10, p. 707).

Hamming, R. W. (1950). Error detecting and error correcting codes. Bell System technical journal, 29(2), 147-160.

See Also

analyse.barcodes

Examples

```
barcodes <- c("AGGT", "TTCC", "CTGA", "GCAA")
barcode.set.distances(barcodes)
barcode.set.distances(barcodes,metric="seqlev")
```

`create.dnabarcodes` *Create a set of DNA barcodes using one of several heuristic methods.*

Description

Creates a DNA barcode set that has error correction and detection properties. The function uses one of four different heuristics to generate the set (clique, conway, sampling, and ashlock) and one of three different distance metrics between individual barcodes (hamming, seqlev, and levenshtein).

For heuristics, inexperienced users should try "conway" and then "ashlock".

The Hamming Distance (`metric="hamming"`) allows the correction/detection of substitutions. The Sequence Levenshtein distance (`metric="seqlev"`) allows the correction/detection of insertions, deletions, and substitutions in barcodes in DNA context. The Levenshtein distance should not be used except by experienced users (see Details).

The functions `create.dnabarcodes.conway`, `create.dnabarcodes.clique`, `create.dnabarcodes.sampling`, and `create.dnabarcodes.ashlock` provide a shortcut to generating barcode sets based on one of the available heuristics and use better default parameters for some.

Usage

```
create.dnabarcodes(n, dist=3, metric=c("hamming","seqlev","levenshtein", "phaseshift"),
                   heuristic=c("conway", "clique", "sampling", "ashlock"),
            filter.triplets=TRUE, filter.gc=TRUE, filter.self_complementary=TRUE,
                   pool = character(), iterations=100, population=200,
                   cores=detectCores()/2, use_cache = FALSE,
                   cost_sub = 1, cost_indel = 1)

create.dnabarcodes.conway(n, heuristic="conway", ...)

create.dnabarcodes.clique(n, heuristic="clique", ...)

create.dnabarcodes.sampling(n, heuristic="sampling", iterations=20000, ...)

create.dnabarcodes.ashlock(n, heuristic="ashlock", iterations=100, population=200, ...)
```

Arguments

`n`	The length of the barcodes (should be smaller than 14). No default.
`dist`	The minimal distance between barcodes that shall be kept. Default is 3.
`metric`	The distance metric that is used to calculate and keep the distance between barcodes. Default is "hamming"

`heuristic` The heuristic algorithm to generate the barcode set. Available are "conway", "clique", "sampling", and "ashlock". The default is "conway".

`filter.triplets` Should sequences that contain at least three repeated equal bases (e.g., AAA, TTT, CCC, or GGG) be filtered out?

`filter.gc` Should sequences that have an unbalanced ratio of bases G or C versus A or T be filtered out?

`filter.self_complementary` Should self complementary sequences be filtered out?

`pool` An optional set of candidate sequences for the DNA barcode sets. If no pool is given, the maximum possible pool is generated internally in the function according to specifications (length `n`, filtered based on `filter.triplets`, `filter.gc`, `filter.self_complementary`). If the pool is given, only sequences from the pool are used as possible barcodes (but still filtered according to filtering parameters of this function). It is save to leave this option unset.

`iterations` In case of `heuristic = "sampling"`: The number of samples that are tested for maximal size. In case of `heuristic = "ashlock"`: The number of iterations of the genetic algorithm that are conducted. Not used in any other case.

`population` Only used for `heuristic = "ashlock"`: The number of chromosomes of the genetic algorithm that are tested. Note: For `heuristic = "ashlock"`, the number of barcode sets that are tested is population + population/2 * (iterations-1).

`cores` The number of cores (CPUs) that will be used for parallel (openMP) calculations.

`use_cache` Shall the distances between each candidate barcode of the pool be calculated in advance? In many cases this increases speed but needs a lot of memory. When in doubt, set to FALSE.

`cost_sub` The cost weight given to a substitution.

`cost_indel` The cost weight given to insertions and deletions.

`...` Arguments passed on to `create.dnabarcodes`.

Details

Different heuristics produce different results in different time. New users should first try "conway" and then "ashlock".

The heuristics "conway" and "clique" produce fast results but are not nearly as good as the heuristics "sampling" and "ashlock". The clique heuristic is a bit slower and needs more memory than the Conway heuristic because it first constructs a graph representation of the pool.

The heuristic "ashlock" is assumed to produce the best heuristic results after a reasonable number of iterations with a good population size.

Distance metrics are the mathematical fairy dust that make the error correction and detection of the barcode sets possible. Different metrics and different distances allow different error corrections/detections.

A high enough Hamming Distance (`metric = "hamming"`) allows the correction/detection of substitutions. Due to the ignorance of insertions and deletions, any changes to the length of the barcode as well as DNA context are ignored, which makes the Hamming distance a simple choice.

A high enough Sequence Levenshtein distance (`metric = "seqlev"`) allows the correction/detection of insertions, deletions, and substitutions in scenarios where the barcode was attached to a DNA sequence. Your sequence read, coming from a Illumina, Roche, PacBio NGS machine of your choice, should then start with that barcode, followed by the insert or an adapter or some random base calls.

A high enough Levenshtein distance (`metric = "levenshtein"`) allows the correction/detection of insertions, deletions, and substitutions in scenarios where the barcode was not attached anywhere, respective where the exact outline of the barcode is known. This is as far as we know in no current NGS technology the case. Do not use this distance metric, except you know what you are doing.

The number of error corrections/detections of the code depends of the enforced distance `dist`. If all conditions are correct, a barcode set with an enforced distance `dist` can correct `k` errors, if

$$k \leq floor(\frac{dist - 1}{2})$$

The detection of `k` errors is possible, if

$$k \leq dist - 1$$

The advantage of this function over already available barcode sets in the scientific community is the ability to flexibly generate new barcode sets of different properties. For example, the function can use a pre-existing barcode library as a candidate set for a better barcode set. In another example, a higher distance (e.g., `dist = 4`) is used. Such a parameter setting would possibly increase the error detection property of the code as well as the average barcode distance, increasing the probability of guessing a barcode during demultiplexing.

The heuristics for the generation of barcodes are as follows:

The Conway heuristic (`heuristic = "conway"`, named after John Conway) starts with an empty set of barcodes, goes through the list of candidate barcodes (the `pool`) in lexicographical order and adds each candidate barcode to the initial set if the distance if the candidate barcode to each barcode in the intial set is at least `d >= dist`.

The Clique heuristic (`heuristic = "conway"`) first generates a graph representation of the pool. Each barcode in the pool is a node of the graph and two barcodes/nodes are connected undirectionally if their distance is at least `d >= dist`. The barcode set problem is now reduced to finding the maximal clique in this graph. Because that problem is also computationally infeasible, we use the heuristic clique algorithm of Pattabiraman et al.

The sampling heuristic (`heuristic = "sampling"`) extends the principle of the Conway heuristic. Instead of starting with an empty initial set, we generate small random sets of barcodes as initial sets (the so called seeds). Those seeds are then "closed" using the Conway method. The size of the seed is fixed to three barcodes. The number of random seeds is given by the parameter `iterations`, hence that often a Conway closure is calculated.

Finally, the Ashlock heuristic (`heuristic = "ashlock"`, named after Daniel Ashlock) extends the sampling heuristic by adding an evolutionary algorithm. A population of random seeds is generated only for the first iteration. Each seed is then closed using the Conway method. The size of the barcode set after closure defines the *fitness* of that seed. For the next iteration, succesful seeds (with a higher fitness) are cloned and slightly mutated (some barcodes in the seed are replaced with a random new barcode). Those changed seeds are now closed again and their respective fitness calculated. In the first iteration, as many instances of the Conway closure are calculated as there are seeds. In the second round, only half of the seeds (the changed ones) are calculated. Therefore, the total number of calculated Conway closures is `iterations + population/2 * (iterations - 1)`.

Value

A vector of characters, representing the DNA barcode set.

References

Buschmann, T. and Bystrykh, L. V. (2013) Levenshtein error-correcting barcodes for multiplexed DNA sequencing. BMC bioinformatics, 14(1), 272. Available from http://www.biomedcentral.com/1471-2105/14/272.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. In Soviet physics doklady (Vol. 10, p. 707).

Hamming, R. W. (1950). Error detecting and error correcting codes. Bell System technical journal, 29(2), 147-160.

Conway, J. and Sloane, N. (1986) Lexicographic codes: error-correcting codes from game theory. Information Theory, IEEE Transactions on, 32(3), 337-348.

Pattabiraman, B., Patwary, M. M. A., Gebremedhin, A. H., Liao, W. K. and Choudhary, A. (2013) Fast algorithms for the maximum clique problem on massive sparse graphs. In Algorithms and Models for the Web Graph (pp. 156-169). Springer International Publishing.

Ashlock, D., Guo, L. and Qiu, F. (2002) Greedy closure evolutionary algorithms. In Computational Intelligence, Proceedings of the World on Congress on (Vol. 2, pp. 1296-1301). IEEE.

Brouwer, A. E., Shearer, L. B. and Sloane, N. I. A. (1990) A new table of constant weight codes. In IEEE Trans Inform Theory.

See Also

create.pool to get the pool of barcode candidates that are used in this function internally.

Examples

```
# Create barcodes of length 5 and minimal Hamming distance of 3:
# (these barcodes can correct up to 1 substitution mutation)
create.dnabarcodes(5) # 30 barcodes
create.dnabarcodes.ashlock(5) # Up to 48 barcodes

# Create barcodes of length 5 with minimal SeqLev distance of 3:
# (barcodes can correct up to 1 insertion/deletion/substitution
#  in DNA context)
create.dnabarcodes(5, metric="seqlev") # 8 barcodes
create.dnabarcodes.ashlock(5, metric="seqlev") # Up to 13

# Create Seq-Lev-barcodes without filtering
length(create.dnabarcodes.ashlock(5, metric="seqlev", filter.triplets=FALSE, filter.gc=FALSE, filter.self_comple

# Create pool, apply additional (useless) filters, create barcode set
pool <- create.pool(5)
length(pool) # 592
pool <- pool[grep("AT",pool,invert=TRUE)]
length(pool) # 468
create.dnabarcodes(5,pool=pool)
```

`create.pool` *Create a pool of barcode candidates.*

Description

This function creates a vector of sequences of equal length with some filtering applied. These sequences function as the pool of candidate sequences from which an actual DNA barcode set may be constructed, for example by passing the pool as a parameter to the function `create.dnabarcodes`.

Sequences in the pool are constructed as all possible concatenations of n bases C,G,A, and T with some of them filtered out for various technical reasons.

Users of this package usually do not need to use this function as the function `create.dnabarcodes` already creates such a pool internally. However, there are some exceptions: A) The user might want to know the set of barcode candidates from which the final DNA barcode set was generated. B) The user might want apply some additional filtering to the pool before constructing a DNA barcode set.

Usage

```
create.pool(n, filter.triplets=TRUE, filter.gc=TRUE, filter.self_complementary=TRUE, cores=detectCore
```

Arguments

`n`	The length of the sequences in the pool (should be smaller than 20)
`filter.triplets`	Should sequences that contain at least three repeated equal bases (e.g., AAA, TTT, CCC, or GGG) be filtered out?
`filter.gc`	Should sequences that have an unbalanced ratio of bases G or C versus A or T be filtered out?
`filter.self_complementary`	Should self complementary sequences be filtered out?
`cores`	The number of cores (CPUs) that will be used for parallel (openMP) calculations.

Value

A vector of characters representing the pool of barcode candidates.

See Also

`create.dnabarcodes`

Examples

```
create.pool(4)
length(create.pool(5))
length(create.pool(5, filter.triplets=FALSE, filter.gc=FALSE, filter.self_complementary=FALSE))
#
```

```
#
# Create pool, apply additional (useless) filters, create barcode set
pool <- create.pool(5)
length(pool) # 592
pool <- pool[grep("AT",pool,invert=TRUE)]
length(pool) # 468
create.dnabarcodes(5,pool=pool)
```

`demultiplex` *Demultiplex a set of reads.*

Description

The function `demultiplex` takes a set of reads that start with a barcode and assigns those reads to a reference barcode while possibly correcting errors.

The correct metric should be used, with `metric = "hamming"` to correct substitution errors and `metric = "seqlev"` to correct insertion, deletion, and substitution errors.

Usage

```
demultiplex(reads, barcodes, metric=c("hamming","seqlev","levenshtein","phaseshift"), cost_sub = 1, c
```

Arguments

`reads`	The reads coming from your sequencing machines that start with a barcode. For `metric = "seqlev"` please provide some context after the (supposed) barcode, at least as many bases as errors that you want to correct.
`barcodes`	The reference barcodes that you used during library preparation and that you want to correct in your reads.
`metric`	The distance metric to be used to assign reads to reference barcodes.
`cost_sub`	The cost weight given to a substitution.
`cost_indel`	The cost weight given to insertions and deletions.

Details

Reads are matched to their correct reference barcodes by calculating the distances between each read and each reference barcode. The reference barcode with the smallest distance to the read is assumed to be the correct original barcode of that read.

For `metric = "hamming"`, only the first `n` (with `n` being the length of the reference barcodes) bases of the read are used for these comparisons and no bases afterwards. Reads with fewer than `n` bases cannot be matched.

For `metric = "seqlev"`, the whole read is compared with the reference barcodes. The Sequence Levenshtein distance was especially developed for barcodes in DNA context and can cope with ambiguities that stem from changes to the length of the barcode.

The Levenshtein distance (`metric = "levenshtein"`) is largely undefined in DNA context and should be avoided. The Levenshtein distance only works if the length both of the reference barcode and the barcode in the read is known. With possible insertions and deletions, this becomes an unknown. For this reason, we always calculate the Levenshtein distance between the whole read and the whole reference barcode without coping with potential side effects.

Value

A vector of reference barcodes of the same length as the input reads. Each reference barcode is the corrected version of the input barcode.

Note

Do not try to correct errors in barcodes that were not systematically constructed for such a correction. To create such a barcode set, have a look into function `create.dnabarcodes`.

See Also

`create.dnabarcodes`, `analyse.barcodes`

Examples

```
# Define some barcodes and inserts
barcodes <- c("AGGT", "TTCC", "CTGA", "GCAA")
insert <- 'ACGCAGGTTGCATATTTTAGGAAGTGAGGAGGAGGCACGGGCTCGAGCTGCGGCTGGGTCTGGGGCGCGG'

# Choose and mutate a couple of thousand barcodes
used_barcodes <- sample(barcodes,10000,replace=TRUE)
mutated_barcodes <- unlist(lapply(strsplit(used_barcodes,""), function(x) { pos <- sample(1:length(x),1); x[pos]

show(setequal(mutated_barcodes, used_barcodes)) # FALSE

# Construct reads (= barcodes + insert)
reads <- paste(mutated_barcodes, insert, sep='')

# Demultiplex
demultiplexed <- demultiplex(reads,barcodes,metric="hamming")

# Show correctness
show(setequal(demultiplexed, used_barcodes)) # TRUE
```

---

distance *Calculate distance between two barcodes.*

Description

The function calculates the distance between two barcodes. The user may choose one of several distance metrics (`"hamming"`, `"seqlev"`, `"levenshtein"`, `"phaseshift"`).

Usage

```
distance(sequence1, sequence2, metric=c("hamming","seqlev","levenshtein", "phaseshift"), cost_sub=1,
```

Arguments

sequence1	The first sequence (a string)
sequence2	The second sequence (a string)
metric	The distance metric which should be calculated.
cost_sub	The cost weight given to a substitution.
cost_indel	The cost weight given to insertions and deletions.

Value

The distance between the two sequences.

References

Buschmann, T. and Bystrykh, L. V. (2013) Levenshtein error-correcting barcodes for multiplexed DNA sequencing. BMC bioinformatics, 14(1), 272. Available from http://www.biomedcentral.com/1471-2105/14/272.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. In Soviet physics doklady (Vol. 10, p. 707).

Hamming, R. W. (1950). Error detecting and error correcting codes. Bell System technical journal, 29(2), 147-160.

Examples

```
distance("AGGT", "TTCC")
distance("AGGT", "TTCC", metric="seqlev")
```

---

mutatedReads *Mock Set of Mutated Reads*

Description

The mock set of mutated reads consists of DNA barcodes from the supplierSet dataset concatenated with random DNA bases. Each read has been mutated to a certain degree (i.e., single bases have been substituted).

The set is only intended for the man page examples.

Format

A character vector of 10,000 DNA sequences.

`supplierSet` *Mock Set of DNA Barcodes*

Description

The mock set of DNA barcodes represents a set that could come from any sample tag or index supplier. The set has a minimum Hamming distance of 3 and therefore allows the correction of a single substitution.

The set is only intended for the manual examples and should not be used in real experiments.

Format

A character vector of 48 DNA barcodes.

Index

∗ **DNA Barcodes**
DNABarcodes-package, 2
∗ **Demultiplexing**
DNABarcodes-package, 2
∗ **Error Correction**
DNABarcodes-package, 2
∗ **Hamming Distance**
DNABarcodes-package, 2
∗ **Levenshtein Distance**
DNABarcodes-package, 2
∗ **Multiplexing**
DNABarcodes-package, 2
∗ **Next Generation Sequencing**
DNABarcodes-package, 2
∗ **Sample Tags**
DNABarcodes-package, 2
∗ **datasets**
mutatedReads, 13
supplierSet, 14

analyse.barcodes, *2*, 3, *5*, *12*

barcode.set.distances, *2*, *4*, 5

create.dnabarcodes, *2*, *5*, 6, *10*, *12*
create.pool, *9*, 10

demultiplex, *2*, 11
distance, 12
DNABarcodes *(*DNABarcodes-package*)*, 2
DNABarcodes-package, 2

mutatedReads, 13

supplierSet, 14