An Introduction to **PGRdup** Package

Aravind, J.\(^1\), Radhamani, J.\(^1\), Kalyani Srinivasan\(^1\), Ananda Subhash, B.\(^2\), and Tyagi, R. K.\(^1\)

2020-07-26

1. ICAR-National Bureau of Plant Genetic Resources, New Delhi, India.
2. Centre for Development of Advanced Computing, Thiruvananthapuram, Kerala, India.

Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Version History</td>
<td>3</td>
</tr>
<tr>
<td>Installation</td>
<td>4</td>
</tr>
<tr>
<td>Data Format</td>
<td>4</td>
</tr>
<tr>
<td>Data Pre-processing</td>
<td>5</td>
</tr>
<tr>
<td>Generation of KWIC Index</td>
<td>9</td>
</tr>
<tr>
<td>Retrieval of Probable Duplicate Sets</td>
<td>13</td>
</tr>
<tr>
<td>Methods</td>
<td>13</td>
</tr>
<tr>
<td>Matching Strategies</td>
<td>16</td>
</tr>
<tr>
<td>Memory and Speed Constraints</td>
<td>20</td>
</tr>
<tr>
<td>Set Review, Modification and Validation</td>
<td>22</td>
</tr>
<tr>
<td>Other Functions</td>
<td>26</td>
</tr>
<tr>
<td>Citing PGRdup</td>
<td>33</td>
</tr>
<tr>
<td>Session Info</td>
<td>33</td>
</tr>
<tr>
<td>References</td>
<td>34</td>
</tr>
</tbody>
</table>

Introduction

**PGRdup** is an **R** package to facilitate the search for probable/possible duplicate accessions in Plant Genetic Resources (PGR) collections using passport databases. Primarily this package implements a workflow (Fig. 1) designed to fetch groups or sets of germplasm accessions with similar passport data particularly in fields associated with accession names within or across PGR passport databases. It offers a suite of functions for data pre-processing, creation of a searchable Key Word in Context (KWIC) index of keywords associated with accession records and the identification of probable duplicate sets by fuzzy, phonetic and semantic matching of keywords. It also has functions to enable the user to review, modify and validate the probable duplicate sets retrieved.

The goal of this document is to introduce the users to these functions and familiarise them with the workflow intended to fetch probable duplicate sets. This document assumes a basic knowledge of **R** programming language.
The functions in this package are primarily built using the R packages `data.table`, `igraph`, `stringdist` and `stringi`. 
An Introduction to PGRdup Package

Version History

The current version of the package is 0.2.3.6. The previous versions are as follows.

Table 1. Version history of PGRdup R package.

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>2015-04-14</td>
</tr>
<tr>
<td>0.2.1</td>
<td>2015-07-23</td>
</tr>
<tr>
<td>0.2.2</td>
<td>2016-03-05</td>
</tr>
<tr>
<td>0.2.2.1</td>
<td>2016-03-09</td>
</tr>
<tr>
<td>0.2.3</td>
<td>2017-02-01</td>
</tr>
<tr>
<td>0.2.3.1</td>
<td>2017-03-15</td>
</tr>
<tr>
<td>0.2.3.2</td>
<td>2017-08-05</td>
</tr>
<tr>
<td>0.2.3.3</td>
<td>2018-01-13</td>
</tr>
<tr>
<td>0.2.3.4</td>
<td>2019-09-19</td>
</tr>
<tr>
<td>0.2.3.5</td>
<td>2020-02-10</td>
</tr>
</tbody>
</table>

To know detailed history of changes use `news(package='PGRdup')`.

Fig. 1. PGRdup workflow and associated functions
Installation

The package can be installed using the following functions:

```r
# Install from CRAN
install.packages('PGRdup', dependencies=TRUE)
```

Uninstalled dependencies (packages which `PGRdup` depends on viz- `data.table`, `igraph`, `stringdist` and `stringi`) are also installed because of the argument `dependencies=TRUE`.

Then the package can be loaded using the function

```r
library(PGRdup)
```

Data Format

The package is essentially designed to operate on PGR passport data present in a data frame object, with each row holding one record and columns representing the attribute fields. For example, consider the dataset GN1000 supplied along with the package.

```r
library(PGRdup)
```

Welcome to PGRdup version 0.2.3.6

```r
# To know how to use this package type:
browseVignettes(package = 'PGRdup')
for the package vignette.

# To know what's new in this version type:
news(package='PGRdup')
for the NEWS file.

# To cite the methods in the package type:
citation(package='PGRdup')

# To suppress this message use:
suppressPackageStartupMessages(library(PGRdup))
```

```r
# Load the dataset to the environment
data(GN1000)
# Show the class of the object
class(GN1000)
```

[1] "data.frame"

```r
# View the first few records in the data frame
head(GN1000)
```

<table>
<thead>
<tr>
<th></th>
<th>CommonName</th>
<th>BotanicalName</th>
<th>NationalID</th>
<th>CollNo</th>
<th>DonorID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Groundnut</td>
<td>Arachis hypogaea</td>
<td>EC100277</td>
<td>Shulamith/ NRCG-14555</td>
<td>ICG-4709</td>
</tr>
<tr>
<td>2</td>
<td>Groundnut</td>
<td>Arachis hypogaea</td>
<td>EC100280</td>
<td>NC</td>
<td>ICG5288</td>
</tr>
<tr>
<td>3</td>
<td>Groundnut</td>
<td>Arachis hypogaea</td>
<td>EC100281</td>
<td>MALIMBA</td>
<td>ICG5289</td>
</tr>
<tr>
<td>4</td>
<td>Groundnut</td>
<td>Arachis hypogaea</td>
<td>EC100713</td>
<td>EC 100713;</td>
<td>ICG5296</td>
</tr>
<tr>
<td>5</td>
<td>Groundnut</td>
<td>Arachis hypogaea</td>
<td>EC100715</td>
<td>EC 100715</td>
<td>ICG5298</td>
</tr>
<tr>
<td>6</td>
<td>Groundnut</td>
<td>Arachis hypogaea</td>
<td>EC100716</td>
<td></td>
<td>ICG-3150</td>
</tr>
</tbody>
</table>
An Introduction to PGRdup Package

## Data Pre-processing

Data pre-processing is a critical step which can affect the quality of the probable duplicate sets being retrieved. It involves data standardization as well as data cleaning which can be achieved using the functions `DataClean`, `MergeKW`, `MergePrefix` and `MergeSuffix`.

The `DataClean` function can be used to clean the character strings in passport data fields (columns) specified as the input character vector `x` according to the conditions specified in the arguments.

Commas, semicolons and colons which are sometimes used to separate multiple strings or names within the same field can be replaced with a single space using the logical arguments `fix.comma`, `fix.semcol` and `fix.col` respectively.

```r
data(x) <- c("A 14; EC 1697", "U 4-4-28; EC 21078; A 32", "PI 262801:CIAT 9075:GKP 9553/90", "NCAC 16049, PI 261987, RCM 493-3")
```

# Replace ',' and ';' with space
```r
DataClean(x, fix.comma=TRUE, fix.semcol=TRUE, fix.col=TRUE, fix.punct=FALSE, fix.space=TRUE, fix.sep=TRUE, fix.leadzero=FALSE)
```

```r
[1] "A 14 EC 1697"          "U 4-4-28 EC 21078 A 32"
[3] "PI 262801:CIAT 9075:GKP 9553/90" "NCAC 16049, PI 261987, RCM 493-3"
```

Similarly the logical argument `fix.bracket` can be used to replace all brackets including parenthesis, square brackets and curly brackets with space.

```r
x <- c("(NRCG-1738)/(NFG649)", "26-5-1[NRCG-2528]", "Ah 1182 {NRCG-4340}")
```

# Replace parenthesis, brackets and curly brackets with space
```r
DataClean(x, fix.comma=FALSE, fix.bracket=TRUE, fix.punct=FALSE, fix.space=FALSE, fix.sep=FALSE, fix.leadzero=FALSE)
```

```r
```
# Replace parenthesis, square brackets and curly brackets with space

```r
DataClean(x, fix.comma=FALSE, fix.semcol=FALSE, fix.col=FALSE,
          fix.bracket=TRUE,
          fix.punct=FALSE, fix.space=FALSE, fix.sep=FALSE, fix.leadzero=FALSE)
```

1. "NRCG-1738 / NFG649" "26-5-1 NRCG-2528" "AH 1182 NRCG-4340"

The logical argument `fix.punct` can be used to remove all punctuation from the data.

```r
x <- c("#26-6-3-1", "Culture No. 857", "U/4/47/13")
x
```

1. "#26-6-3-1" "Culture No. 857" "U/4/47/13"

# Remove punctuation

```r
DataClean(x, fix.comma=FALSE, fix.semcol=FALSE, fix.col=FALSE,
          fix.bracket=FALSE,
          fix.punct=TRUE,
          fix.space=FALSE, fix.sep=FALSE, fix.leadzero=FALSE)
```

1. "26631" "CULTURE NO 857" "U44713"

`fix.space` can be used to convert all space characters such as tab, newline, vertical tab, form feed and carriage return to spaces and finally convert multiple spaces to single space.

```r
x <- c("RS 1", "GKSPScGb 208 PI 475855")
x
```

1. "RS 1" "GKSPSCGb 208 PI 475855"

# Replace all space characters to space and convert multiple spaces to single space

```r
DataClean(x, fix.comma=FALSE, fix.semcol=FALSE, fix.col=FALSE,
          fix.bracket=FALSE,
          fix.punct=FALSE,
          fix.space=TRUE, fix.sep=FALSE, fix.leadzero=FALSE)
```

1. "RS 1" "GKSPSCGb 208 PI 475855"

`fix.leadzero` can be used to remove leading zeros from accession name fields to facilitate matching to identify probable duplicates.

```r
x <- c("EC 0016664", "EC0001690")
x
```

1. "EC 0016664" "EC0001690"
An Introduction to PGRdup Package

Data Pre-processing

# Remove leading zeros
DataClean(x, fix.comma=FALSE, fix.semcol=FALSE, fix.col=FALSE, fix.bracket=FALSE, fix.punct=FALSE, fix.space=FALSE, fix.sep=FALSE, fix.leadzero=TRUE)

[1] "EC 16664" "EC1690"

This function can hence be made use of in tidying up multiple forms of messy data existing in fields associated with accession names in PGR passport databases (Table 1).

names <- c("S7-12-6", "ICG-3505", "U 4-47-18;EC 21127", "AH 6481", "RS 1", "AK 12-24", "2-5 (NRCG-4053)", "T78, Mwitunde", "ICG 3410", "#648-4 (Gwalior)", "TG4;U/4/47/13", "EC0021003")

Table 2. Data pre-processing using DataClean.

<table>
<thead>
<tr>
<th>names</th>
<th>DataClean(names)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S7-12-6</td>
<td>S7126</td>
</tr>
<tr>
<td>ICG-3505</td>
<td>ICG3505</td>
</tr>
<tr>
<td>U 4-47-18;EC 21127</td>
<td>U44718 EC21127</td>
</tr>
<tr>
<td>AH 6481</td>
<td>AH6481</td>
</tr>
<tr>
<td>RS 1</td>
<td>RS1</td>
</tr>
<tr>
<td>AK 12-24</td>
<td>AK1224</td>
</tr>
<tr>
<td>2-5 (NRCG-4053)</td>
<td>25 NRCG4053</td>
</tr>
<tr>
<td>T78, Mwitunde</td>
<td>T78 MWITUNDE</td>
</tr>
<tr>
<td>ICG 3410</td>
<td>ICG3410</td>
</tr>
<tr>
<td>#648-4 (Gwalior)</td>
<td>6484 GWALIOR</td>
</tr>
<tr>
<td>TG4;U/4/47/13</td>
<td>TG4 U44713</td>
</tr>
<tr>
<td>EC0021003</td>
<td>EC21003</td>
</tr>
</tbody>
</table>

Several common keyword string pairs or keyword prefixes and suffixes exist in fields associated with accession names in PGR passport databases. They can be merged using the functions MergeKW, MergePrefix and MergeSuffix respectively. The keyword string pairs, prefixes and suffixes can be supplied as a list or a vector to the argument y in these functions.


Table 2. Data pre-processing using DataClean.

[1] "Punjab Bold" "Gujarat- Dwarf" "Nagpur.local"
An Introduction to PGRdup Package

Data Pre-processing

[4] "SAM COL 144" "SAM COL--280" "NIZAMABAD-LOCAL"
[7] "Dark Green Mutant" "Dixie-Giant" "Georgia- Bunch"
[10] "Uganda-erect" "Small Japan" "Castle Cary"
[13] "Punjab erect" "Improved small japan" "Dark Purple"

# Merge pairs of strings
y1 <- list(c("Gujarat", "Dwarf"), c("Castle", "Cary"), c("Small", "Japan"),
           c("Big", "Japan"), c("Mani", "Blanco"), c("Uganda", "Erect"),
           c("Mota", "Company"))
names <- MergeKW(names, y1, delim = c("space", "dash", "period"))

# Merge prefix strings
y2 <- c("Light", "Small", "Improved", "Punjab", "SAM", "Dark")
names <- MergePrefix(names, y2, delim = c("space", "dash", "period"))

# Merge suffix strings
y3 <- c("Local", "Bold", "Cary", "Mutant", "Runner", "Giant", "No.",
        "Bunch", "Peanut")
names <- MergeSuffix(names, y3, delim = c("space", "dash", "period"))

These functions can be applied over multiple columns(fields) in a data frame using the lapply function.

# Load example dataset
GN <- GN1000

# Specify as a vector the database fields to be used
GNfields <- c("NationalID", "CollNo", "DonorID", "OtherID1", "OtherID2")

head(GN[GNfields])

# Clean the data
GN[GNfields] <- lapply(GN[GNfields], function(x) DataClean(x))
y1 <- list(c("Gujarat", "Dwarf"), c("Castle", "Cary"), c("Small", "Japan"),
           c("Big", "Japan"), c("Mani", "Blanco"), c("Uganda", "Erect"),
           c("Mota", "Company"))
y2 <- c("Dark", "Light", "Small", "Improved", "Punjab", "SAM")
y3 <- c("Local", "Bold", "Cary", "Mutant", "Runner", "Giant", "No.",
        "Bunch", "Peanut")

GN[GNfields] <- lapply(GN[GNfields],
          function(x) MergeKW(x, y1, delim = c("space", "dash")))
GN[GNfields] <- lapply(GN[GNfields],
          function(x) MergePrefix(x, y2, delim = c("space", "dash")))
Generation of KWIC Index

The function `KWIC` generates a Key Word in Context index (Knüpffer 1988; Knüpffer, Frese, and Jongen 1997) from the data frame of a PGR passport database based on the fields(columns) specified in the argument `fields` along with the keyword frequencies and gives the output as a list of class `KWIC`. The first element of the vector specified in `fields` is considered as the primary key or identifier which uniquely identifies all rows in the data frame.

This function fetches keywords from different fields specified, which can be subsequently used for matching to identify probable duplicates. The frequencies of the keywords retrieved can help in determining if further data pre-processing is required and also to decide whether any common keywords can be exempted from matching (Fig. 2).

```r
# Load example dataset
GN <- GN1000

# Specify as a vector the database fields to be used
GNfields <- c("NationalID", "CollNo", "DonorID", "OtherID1", "OtherID2")

# Clean the data
GN[GNfields] <- lapply(GN[GNfields], function(x) DataClean(x))
y1 <- list(c("Gujarat", "Dwarf"), c("Castle", "Cary"), c("Small", "Japan"),
c("Big", "Japan"), c("Mani", "Blanco"), c("Uganda", "Erect"),
c("Hota", "Company"))
y2 <- c("Dark", "Light", "Small", "Improved", "Punjab", "SAM")
y3 <- c("Local", "Bold", "Cary", "Mutant", "Runner", "Giant", "No.",
"Bunch", "Peanut")
GN[GNfields] <- lapply(GN[GNfields],
function(x) MergeKW(x, y1, delim = c("space", "dash")))
GN[GNfields] <- lapply(GN[GNfields],
function(x) MergePrefix(x, y2, delim = c("space", "dash")))
GN[GNfields] <- lapply(GN[GNfields],
function(x) MergeSuffix(x, y3, delim = c("space", "dash")))

# Generate the KWIC index
GNKWIC <- KWIC(GN, GNfields, min.freq = 1)
class(GNKWIC)
```

```
[1] "KWIC"

GNKWIC

KWIC fields : NationalID CollNo DonorID OtherID1 OtherID2
Number of keywords : 3893
```
Number of distinct keywords : 3109

# Retrieve the KWIC index from the KWIC object
KWIC <- GNKWIC[[1]]
KWIC <- KWIC[order(KWIC$KEYWORD, decreasing = TRUE),]
head(KWIC[,c("PRIM_ID", "KWIC_L", "KWIC_KW", "KWIC_R")], n = 10)

## KWIC_L KWIC_KW
## PRIM_ID KWIC_R
## 550 EC490380 = = ICG1122 = = LIN YUCH
## 435 EC36893 = EC36893 = YUAN
## 434 EC36893 = EC36893 = YUAN YOUNG
## 1287 EC613524 = NRCG9225 = = PEI KANGPE YOUDON
## 1703 IC113088 = = = SB XI
## 1741 IC296965 IC296965 = SB X11 X V11 = ICG1769 = = SB XI
## 3385 IC445197 IC445197 = X144B28
## 3483 IC494754 IC494754 = IC494754 = = IC7686 = = X144B28
## 2090 IC304018 IC304018 = 144B19B NRCG = ICG1561 = = X144B19B
## 1735 IC296965 IC296965 = SB X11

# Retrieve the keyword frequencies from the KWIC object
KeywordFreq <- GNKWIC[[2]]
head(KeywordFreq)

## Keyword Freq
## 1 OVERO 25
## 2 S1 19
## 3 A 11
## 4 RED 11
## 5 OVER 10
## 6 PURPLE 10
Fig. 2. Word cloud of keywords retrieved

The function will throw an error in case of duplicates or NULL values in the primary key/ID field mentioned.

```r
GN <- GN1000
GN[GNfields] <- lapply(GN[GNfields], function(x) DataClean(x))
# Generate dummy duplicates for illustration
GN[1001:1005,] <- GN[1:5,]
# Generate dummy NULL values for illustration
GN[1001,3] <- ""
GN[1002,3] <- ""
GN[1001:1005,]

CommonName BotanicalName NationalID CollNo DonorID
1001 Groundnut Arachis hypogaea SHULAMITH NRCG14555 ICG4709
1002 Groundnut Arachis hypogaea NC ICG5288
1003 Groundnut Arachis hypogaea EC100281 MALIMBA ICG5289
1004 Groundnut Arachis hypogaea EC100713 EC100713 ICG5296
1005 Groundnut Arachis hypogaea EC100715 EC100715 ICG5298

OtherID1 OtherID2 BioStatus SourceCountry TransferYear
1001 U44712 Landrace Israel 2014
1002 NCS NC5 Landrace United States of America 2004
1003 EC100281 Landrace Malawi 2004
1004 STARR Landrace United States of America 2004
1005 COMET Landrace United States of America 2004

GNKWIC <- KWIC(GN, GNfields, min.freq=1)

Error in KWIC(GN, GNfields, min.freq = 1) :
  Primary key/ID field should be unique and not NULL
Use PGRdup::ValidatePrimKey() to identify and rectify the aberrant records first
```
The erroneous records can be identified using the helper function `ValidatePrimKey`.

```r
# Validate the primary key/ID field for duplication or existence of NULL values
ValidatePrimKey(x = GN, prim.key = "NationalID")
```

```
$message1
[1] "ERROR: Duplicated records found in prim.key field"
```

```
$Duplicates
     CommonName BotanicalName NationalID CollNo DonorID
1001 Groundnut Arachis hypogaea SHULAMITH NRCG14555 ICG4709
1002 Groundnut Arachis hypogaea NG ICG5288
1003 Groundnut Arachis hypogaea EC100281 MALIMBA ICG5289
1004 Groundnut Arachis hypogaea EC100713 EC100713 ICG5296
1005 Groundnut Arachis hypogaea EC100715 EC100715 ICG5298
```

```
$NullRecords
     CommonName BotanicalName NationalID CollNo DonorID
1001 Groundnut Arachis hypogaea SHULAMITH NRCG14555 ICG4709
1002 Groundnut Arachis hypogaea NC ICG5288
```

```
# Remove the offending records
GN <- GN[-c(1001:1005),]
# Validate again
ValidatePrimKey(x = GN, prim.key = "NationalID")
```

```
$message1
[1] "OK: No duplicated records found in prim.key field"
```

```
$Duplicates
NULL
```

```
$message2
[1] "OK: No NULL records found in prim.key field"
```

```
$NullRecords
NULL
```
Retrieval of Probable Duplicate Sets

Once KWIC indexes are generated, probable duplicates of germplasm accessions can be identified by fuzzy, phonetic and semantic matching of the associated keywords using the function `ProbDup`. The sets are retrieved as a list of data frames of class `ProbDup`.

Keywords that are not to be used for matching can be specified as a vector in the `excep` argument.

Methods

The function can execute matching according to either one of the following three methods as specified by the `method` argument.

1. **Method "a"** : Performs string matching of keywords in a single KWIC index to identify probable duplicates of accessions in a single PGR passport database.

```r
# Load example dataset
GN <- GN1000

# Specify as a vector the database fields to be used
GNfields <- c("NationalID", "CollNo", "DonorID", "OtherID1", "OtherID2")

# Clean the data
GN[GNfields] <- lapply(GN[GNfields], function(x) DataClean(x))

y1 <- list(c("Gujarat", "Dwarf"), c("Castle", "Cary"), c("Small", "Japan"), c("Big", "Japan"), c("Mani", "Blanco"), c("Uganda", "Erect"), c("Mota", "Company"))
y2 <- c("Dark", "Light", "Small", "Improved", "Punjab", "SAM")
y3 <- c("Local", "Bold", "Cary", "Mutant", "Runner", "Giant", "No.", "Bunch", "Peanut")

GN[GNfields] <- lapply(GN[GNfields], function(x) MergeKW(x, y1, delim = c("space", "dash")))
GN[GNfields] <- lapply(GN[GNfields], function(x) MergePrefix(x, y2, delim = c("space", "dash")))
GN[GNfields] <- lapply(GN[GNfields], function(x) MergeSuffix(x, y3, delim = c("space", "dash")))

# Generate the KWIC index
GNKWIC <- KWIC(GN, GNfields)

# Specify the exceptions as a vector

# Fetch fuzzy duplicates by method 'a'
GNdup <- ProbDup(kwic1 = GNKWIC, method = "a", excep = exep, fuzzy = TRUE, phonetic = FALSE, semantic = FALSE)

Fuzzy matching

<table>
<thead>
<tr>
<th>class(GNdup)</th>
</tr>
</thead>
</table>
An Introduction to **PGRdup** Package

**Retrieval of Probable Duplicate Sets**

[1] "ProbDup"

**GNdup**

Method : a

**KWIC1 fields : NationalID CollNo DonorID OtherID1 OtherID2**

<table>
<thead>
<tr>
<th>No.of.Sets</th>
<th>No.of.Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>FuzzyDuplicates</td>
<td>378 745</td>
</tr>
<tr>
<td>Total</td>
<td>378 745(Distinct:745)</td>
</tr>
</tbody>
</table>

# Fetch phonetic duplicates by method 'a'

```r
GNdup <- ProbDup(kwic1 = GNKWIC, method = "a", excep = exep, fuzzy = FALSE, phonetic = TRUE, semantic = FALSE)
```

**Phonetic matching**

<table>
<thead>
<tr>
<th>class(GNdup)</th>
</tr>
</thead>
<tbody>
<tr>
<td>=============</td>
</tr>
</tbody>
</table>

[1] "ProbDup"

**GNdup**

Method : a

**KWIC1 fields : NationalID CollNo DonorID OtherID1 OtherID2**

<table>
<thead>
<tr>
<th>No.of.Sets</th>
<th>No.of.Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhoneticDuplicates</td>
<td>99 260</td>
</tr>
<tr>
<td>Total</td>
<td>99 260(Distinct:260)</td>
</tr>
</tbody>
</table>

2. **Method "b"**: Performs string matching of keywords in the first KWIC index (query) with that of the keywords in the second index (source) to identify probable duplicates of accessions of the first PGR passport database among the accessions in the second database.

3. **Method "c"**: Performs string matching of keywords in two different KWIC indexes jointly to identify probable duplicates of accessions from among two PGR passport databases.

**# Load PGR passport databases**

```r
GN1 <- GN1000[grepl("^ICG", GN1000$DonorID),]
GN1$DonorID <- NULL
GN2 <- GN1000[grepl("^ICG", GN1000$DonorID),]
GN2$NationalID <- NULL
```

**# Specify database fields to use**

```r
GN1fields <- c("NationalID", "CollNo", "OtherID1", "OtherID2")
GN2fields <- c("DonorID", "CollNo", "OtherID1", "OtherID2")
```

**# Clean the data**

```r
y1 <- list(c("Gujarat", "Dwarf"), c("Castle", "Cary"), c("Small", "Japan"), c("Big", "Japan"), c("Mani", "Blanco"), c("Uganda", "Erect"), c("Mota", "Company"))
y2 <- c("Dark", "Light", "Small", "Improved", "Punjab", "SAM")
y3 <- c("Local", "Bold", "Cary", "Mutant", "Runner", "Giant", "No.")
```
An Introduction to \texttt{PGRdup} Package

Retrieval of Probable Duplicate Sets

```

"Bunch", "Peanut")
GN1[GN1fields] <- \texttt{lapply}(GN1[GN1fields],
  \texttt{function}(x) \texttt{MergeKW}(x, y1, \texttt{delim} = \texttt{c}("space", "dash")))
GN1[GN1fields] <- \texttt{lapply}(GN1[GN1fields],
  \texttt{function}(x) \texttt{MergePrefix}(x, y2, \texttt{delim} = \texttt{c}("space", "dash")))
GN1[GN1fields] <- \texttt{lapply}(GN1[GN1fields],
  \texttt{function}(x) \texttt{MergeSuffix}(x, y3, \texttt{delim} = \texttt{c}("space", "dash")))

GN2[GN2fields] <- \texttt{lapply}(GN2[GN2fields],
  \texttt{function}(x) \texttt{MergeKW}(x, y1, \texttt{delim} = \texttt{c}("space", "dash")))
GN2[GN2fields] <- \texttt{lapply}(GN2[GN2fields],
  \texttt{function}(x) \texttt{MergePrefix}(x, y2, \texttt{delim} = \texttt{c}("space", "dash")))
GN2[GN2fields] <- \texttt{lapply}(GN2[GN2fields],
  \texttt{function}(x) \texttt{MergeSuffix}(x, y3, \texttt{delim} = \texttt{c}("space", "dash")))

# Remove duplicated DonorID records in GN2
GN2 <- GN2[\texttt{!duplicated}(GN2$DonorID), ]

# Generate KWIC index
GN1KWIC <- \texttt{KWIC}(GN1, GN1fields)
GN2KWIC <- \texttt{KWIC}(GN2, GN2fields)

# Specify the exceptions as a vector
exep <- \texttt{c}("A", "B", "BIG", "BOLD", "BUNCH", "C", "COMPANY", "CULTURE",
  "DARK", "E", "EARLY", "EC", "ERECT", "EXOTIC", "FLESH", "GROUNDNUT",
  "GUTHUKAI", "IMPROVED", "K", "KUTHUKAIDAL", "KUTHUKAI", "LARGE",
  "RED", "RUNNER", "S1", "SAM", "SMALL", "SPANISH", "TAN", "TYPE",
  "U", "VALENCIA", "VIRGINIA", "WHITE")

# Fetch fuzzy and phonetic duplicate sets by method b
GNdupb <- \texttt{ProbDup}(kwic1 = GN1KWIC, kwic2 = GN2KWIC, \texttt{method} = "b",
  \texttt{excep} = exep, \texttt{fuzzy} = \texttt{TRUE}, \texttt{phonetic} = \texttt{TRUE},
  \texttt{encoding} = "primary", \texttt{semantic} = \texttt{FALSE})

Fuzzy matching

Phonetic matching

\texttt{class}(GNdupb)

[1] "ProbDup"

GNdupb

Method : b

KWIC1 fields : NationalID CollNo OtherID1 OtherID2

KWIC2 fields : DonorID CollNo OtherID1 OtherID2

\begin{tabular}{|l|c|c|}
\hline
\textbf{No..of.Sets} & \textbf{No..of.Records} \\
\hline
FuzzyDuplicates & 107 & 353 \\
PhoneticDuplicates & 41 & 126 \\
\hline
\end{tabular}

15
An Introduction to PGRdup Package

Retrieval of Probable Duplicate Sets

Total 148 479 (Distinct:383)

Fuzzy matching

Phonetic matching

`class(GNdupc)`

[1] "ProbDup"

Method: c

KWIC1 fields: NationalID CollNo OtherID1 OtherID2

KWIC2 fields: DonorID CollNo OtherID1 OtherID2

<table>
<thead>
<tr>
<th>No..of.Sets</th>
<th>No..of.Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>FuzzyDuplicates</td>
<td>363</td>
</tr>
<tr>
<td>PhoneticDuplicates</td>
<td>98</td>
</tr>
<tr>
<td>Total</td>
<td>461</td>
</tr>
</tbody>
</table>

Matching Strategies

1. **Fuzzy matching** or approximate string matching of keywords is carried out by computing the generalized levenshtein (edit) distance between them. This distance measure counts the number of deletions, insertions and substitutions necessary to turn one string to another.

   ```r
   # Load example dataset
   GN <- GN1000
   
   # Specify as a vector the database fields to be used
   GNfields <- c("NationalID", "CollNo", "DonorID", "OtherID1", "OtherID2")
   
   # Clean the data
   GN[GNfields] <- lapply(GN[GNfields], function(x) DataClean(x))
   y1 <- list(c("Gujarat", "Dwarf"), c("Castle", "Cary"), c("Small", "Japan"), c("Big", "Japan"), c("Mani", "Blanco"), c("Uganda", "Erect"), c("Mota", "Company"))
   y2 <- c("Dark", "Light", "Small", "Improved", "Punjab", "SAM")
   y3 <- c("Local", "Bold", "Cary", "Mutant", "Runner", "Giant", "No.", "Bunch", "Peanut")
   GN[GNfields] <- lapply(GN[GNfields], function(x) MergeKW(x, y1, delim = c("space", "dash")))
   GN[GNfields] <- lapply(GN[GNfields], function(x) MergePrefix(x, y2, delim = c("space", "dash")))
   GN[GNfields] <- lapply(GN[GNfields], function(x) MergeSuffix(x, y3, delim = c("space", "dash")))
   ```
# Generate the KWIC index

GNKWIC <- KWIC(GN, GNfields)

# Specify the exceptions as a vector

          "DARK", "E", "EARLY", "EC", "ERECT", "EXOTIC", "FLESH", "GROUNDNUT",
          "GUTHUKAI", "IMPROVED", "K", "KUTHUKADAL", "KUTHUKAI", "LARGE",
          "RED", "RUNNER", "S1", "SAM", "SMALL", "SPANISH", "TAN", "TYPE",
          "U", "VALENCIA", "VIRGINIA", "WHITE")

# Fetch fuzzy duplicates

GNdup <- ProbDup(kwic1 = GNKWIC, method = "a", excep = exep,
                 fuzzy = TRUE, max.dist = 3,
                 phonetic = FALSE, semantic = FALSE)

Fuzzy matching

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GNdup</td>
</tr>
</tbody>
</table>

Method : a

KWIC1 fields : NationalID CollNo DonorID OtherID1 OtherID2

<table>
<thead>
<tr>
<th>No.of.Sets</th>
<th>No.of.Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>FuzzyDuplicates</td>
<td>378 745</td>
</tr>
<tr>
<td>Total</td>
<td>378 745(Distinct:745)</td>
</tr>
</tbody>
</table>

The maximum distance to be considered for a match can be specified by max.dist argument.

GNdup <- ProbDup(kwic1 = GNKWIC, method = "a", excep = exep,
                 fuzzy = TRUE, max.dist = 1,
                 phonetic = FALSE, semantic = FALSE)

Fuzzy matching

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GNdup</td>
</tr>
</tbody>
</table>

Method : a

KWIC1 fields : NationalID CollNo DonorID OtherID1 OtherID2

<table>
<thead>
<tr>
<th>No.of.Sets</th>
<th>No.of.Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>FuzzyDuplicates</td>
<td>288 679</td>
</tr>
<tr>
<td>Total</td>
<td>288 679(Distinct:679)</td>
</tr>
</tbody>
</table>

Exact matching can be enforced with the argument force.exact set as TRUE. It can be used to avoid fuzzy matching when the number of alphabet characters in keywords is lesser than a critical value (max.alpha). Similarly, the value of max.digit can also be set according to the requirements to enforce exact matching. The default value of Inf avoids fuzzy matching and enforces exact matching for all keywords having any numerical characters. If max.digit and max.alpha are both set to Inf, exact matching will be enforced for all the keywords.

When exact matching is enforced, for keywords having both alphabet and numeric characters and with the number of alphabet characters greater than max.digit, matching will be carried out separately for alphabet
and numeric characters present.

\[
\text{GNdup} \leftarrow \text{ProbDup}(\text{kwic1} = \text{GNKWIC, method = "a", excep = exep, fuzzy = TRUE, force.exact = TRUE, max.alpha = 4, max.digit = Inf, phonetic = FALSE, semantic = FALSE})
\]

Fuzzy matching

| |-------------------
| GNdup
Method : a

KWIC1 fields : NationalID CollNo DonorID OtherID1 OtherID2

\[
\text{No.of.Sets} \quad \text{No.of.Records}
\]

FuzzyDuplicates 378 745
Total 378 745(Distinct:745)

2. Phonetic matching of keywords is carried out using the Double Metaphone phonetic algorithm which is implemented as the helper function \text{DoubleMetaphone}, (Philips 2000), to identify keywords that have the similar pronunciation.

\[
\text{GNdup} \leftarrow \text{ProbDup}(\text{kwic1} = \text{GNKWIC, method = "a", excep = exep, fuzzy = FALSE, phonetic = TRUE, semantic = FALSE})
\]

Phonetic matching

| |-------------------
| GNdup
Method : a

KWIC1 fields : NationalID CollNo DonorID OtherID1 OtherID2

\[
\text{No.of.Sets} \quad \text{No.of.Records}
\]

PhoneticDuplicates 99 260
Total 99 260(Distinct:260)

Either the primary or alternate encodings can be used by specifying the \text{encoding} argument.

\[
\text{GNdup} \leftarrow \text{ProbDup}(\text{kwic1} = \text{GNKWIC, method = "a", excep = exep, fuzzy = FALSE, phonetic = TRUE, encoding = "alternate", semantic = FALSE})
\]

Phonetic matching

| |-------------------
| GNdup
Method : a

KWIC1 fields : NationalID CollNo DonorID OtherID1 OtherID2

\[
\text{No.of.Sets} \quad \text{No.of.Records}
\]
An Introduction to PGRdup Package

Retrieval of Probable Duplicate Sets

PhoneticDuplicates 98 263
Total 98 263(Distinct:263)

The argument phon.min.alpha sets the limits for the number of alphabet characters to be present in a string for executing phonetic matching.

```r
GNdup <- ProbDup(kwic1 = GNKWIC, method = "a", excep = exep,
                 fuzzy = FALSE,
                 phonetic = TRUE, encoding = "alternate", phon.min.alpha = 4,
                 semantic = FALSE)
```

Phonetic matching

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GNdup</td>
</tr>
<tr>
<td>Method : a</td>
</tr>
</tbody>
</table>

KWIC1 fields : NationalID CollNo DonorID OtherID1 OtherID2

<table>
<thead>
<tr>
<th>No..of.Sets</th>
<th>No..of.Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhoneticDuplicates</td>
<td>304 451</td>
</tr>
<tr>
<td>Total</td>
<td>304 451(Distinct:451)</td>
</tr>
</tbody>
</table>

Similarly min.enc sets the limits for the number of characters to be present in the encoding of a keyword for phonetic matching.

```r
GNdup <- ProbDup(kwic1 = GNKWIC, method = "a", excep = exep,
                 fuzzy = FALSE,
                 phonetic = TRUE, encoding = "alternate", min.enc = 4,
                 semantic = FALSE)
```

Phonetic matching

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GNdup</td>
</tr>
<tr>
<td>Method : a</td>
</tr>
</tbody>
</table>

KWIC1 fields : NationalID CollNo DonorID OtherID1 OtherID2

<table>
<thead>
<tr>
<th>No..of.Sets</th>
<th>No..of.Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhoneticDuplicates</td>
<td>59 156</td>
</tr>
<tr>
<td>Total</td>
<td>59 156(Distinct:156)</td>
</tr>
</tbody>
</table>

3. Semantic matching matches keywords based on a list of accession name synonyms supplied as list with character vectors of synonym sets (synsets) to the syn argument. Synonyms in this context refer to interchangeable identifiers or names by which an accession is recognized. Multiple keywords specified as members of the same synset in syn are matched. To facilitate accurate identification of synonyms from the KWIC index, identical data standardization operations using the Merge* and DataClean functions for both the original database fields and the synset list are recommended.

```r
# Specify the synsets as a list
syn <- list(c("CHANDRA", "AH 114"), c("TG-1", "VIKRAM"))

# Clean the data in the synsets
syn <- lapply(syn, DataClean)
```
An Introduction to PGRdup Package

Retrieval of Probable Duplicate Sets

GNdup <- ProbDup(kwic = GNKWIC, method = "a", excep = exep,
                 fuzzy = FALSE, phonetic = FALSE,
                 semantic = TRUE, syn = syn)

Semantic matching

<table>
<thead>
<tr>
<th>GNdup</th>
</tr>
</thead>
</table>
Method : a

KWIC1 fields : NationalID CollNo DonorID OtherID1 OtherID2

<table>
<thead>
<tr>
<th>No. of Sets</th>
<th>No. of Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>SemanticDuplicates</td>
<td>2 5</td>
</tr>
<tr>
<td>Total</td>
<td>2 5 (Distinct: 5)</td>
</tr>
</tbody>
</table>

Memory and Speed Constraints

As the number of keywords in the KWIC indexes increases, the memory consumption by the function also increases proportionally. This is due to the reason that for string matching, this function relies upon creation of an $n \times m$ matrix of all possible keyword pairs for comparison, where $n$ and $m$ are the number of keywords in the query and source indexes respectively. This can lead to cannot allocate vector of size... errors in case of large KWIC indexes where the comparison matrix is too large to reside in memory. In such a case, the chunksize argument can be reduced from the default 1000 to get the appropriate size of the KWIC index keyword block to be used for searching for matches at a time. However a smaller chunksize may lead to longer computation time due to the memory-time trade-off.

The progress of matching is displayed in the console as number of keyword blocks completed out of the total number of blocks, the percentage of achievement and a text-based progress bar.

In case of multi-byte characters in keywords, the speed of keyword matching is further dependent upon the useBytes argument as described in help("stringdist-encoding") for the stringdist function in the namesake package (van der Loo 2014), which is made use of here for string matching.

The CPU time taken for retrieval of probable duplicate sets under different options for the arguments chunksize and useBytes can be visualized using the microbenchmark package (Fig. 3).

# Load example dataset
GN <- GN1000

# Specify as a vector the database fields to be used
GNfields <- c("NationalID", "CollNo", "DonorID", "OtherID1", "OtherID2")

# Clean the data
GN[GNfields] <- lapply(GN[GNfields], function(x) DataClean(x))
y1 <- list(c("Gujarat", "Dwarf"), c("Castle", "Cary"), c("Small", "Japan"),
          c("Big", "Japan"), c("Mani", "Blanco"), c("Uganda", "Erect"),
          c("Mota", "Company"))
y2 <- c("Dark", "Light", "Small", "Improved", "Punjab", "SAM")
y3 <- c("Local", "Bold", "Cary", "Mutant", "Runner", "Giant", "No.", "Bunch", "Peanut")
GN[GNfields] <- lapply(GN[GNfields], function(x) MergeKW(x, y1, delim = c("space", "dash")))
GN[GNfields] <- lapply(GN[GNfields], function(x) MergePrefix(x, y2, delim = c("space", "dash")))
GN[GNfields] <- lapply(GN[GNfields],

function(x) MergeSuffix(x, y3, delim = c("space", "dash")))

# Generate the KWIC index
GNKWIC <- KWIC(GN, GNfields)

# Specify the exceptions as a vector

# Specify the synsets as a list
syn <- list(c("CHANDRA", "AH 114"), c("TG-1", "VIKRAM"))
syn <- lapply(syn, DataClean)

timings <- microbenchmark::microbenchmark(
  # Fetch duplicate sets with default chunk.size
  t1 = ProbDup(kwic1 = GNKWIC, method = "a", excep = exep,
               chunksize = 1000, useBytes = TRUE,
               fuzzy = TRUE, phonetic = TRUE,
               semantic = TRUE, syn = syn),
  # Fetch duplicate sets chunk.size 2000
  t2 = ProbDup(kwic1 = GNKWIC, method = "a", excep = exep,
               chunksize = 2000, useBytes = TRUE,
               fuzzy = TRUE, phonetic = TRUE,
               semantic = TRUE, syn = syn),
  # Fetch duplicate sets chunk.size 100
  t3 = ProbDup(kwic1 = GNKWIC, method = "a", excep = exep,
               chunksize = 100, useBytes = TRUE,
               fuzzy = TRUE, phonetic = TRUE,
               semantic = TRUE, syn = syn),
  # Fetch duplicate sets useBytes = FALSE
  t4 = ProbDup(kwic1 = GNKWIC, method = "a", excep = exep,
               chunksize = 1000, useBytes = FALSE,
               fuzzy = TRUE, phonetic = TRUE,
               semantic = TRUE, syn = syn), times = 10)

plot(timings, col = c("#1B9E77", "#D95F02", "#7570B3", "#E7298A"),
     xlab = "Expression", ylab = "Time")
legend("topright", c("t1 : chunksize = 1000,\n                      useBytes = T (default)\n", "t2 : chunksize = 2000,\n                      useBytes = T\n", "t3 : chunksize = 100,\n                      useBytes = T\n", "t4 : chunksize = 1000,\n                      useBytes = F\n"),
       bty = "n", cex = 0.6)
Set Review, Modification and Validation

The initially retrieved sets may be intersecting with each other because there might be accessions which occur in more than one duplicate set. Disjoint sets can be generated by merging such overlapping sets using the function DisProbDup.

Disjoint sets are retrieved either individually for each type of probable duplicate sets or considering all types of sets simultaneously. In case of the latter, the disjoint of all the types of sets alone are returned in the output as an additional data frame DisjointDuplicates in an object of class ProbDup.

```r
# Load example dataset
GN <- GN1000

# Specify as a vector the database fields to be used
GNfields <- c("NationalID", "CollNo", "DonorID", "OtherID1", "OtherID2")

# Clean the data
GN[GNfields] <- lapply(GN[GNfields], function(x) DataClean(x))
y1 <- list(c("Gujarat", "Dwarf"), c("Castle", "Cary"), c("Small", "Japan"),
          c("Big", "Japan"), c("Mani", "Blanco"), c("Uganda", "Erect"),
          c("Mota", "Company"))
y2 <- c("Dark", "Light", "Small", "Improved", "Punjab", "SAM")
y3 <- c("Local", "Bold", "Cary", "Mutant", "Runner", "Giant", "No.",
       "Bunch", "Peanut")
GN[GNfields] <- lapply(GN[GNfields],
```
function(x) MergeKW(x, y1, delim = c("space", "dash"))

GN[GNfields] <- lapply(GN[GNfields],
  function(x) MergePrefix(x, y2, delim = c("space", "dash")))

GN[GNfields] <- lapply(GN[GNfields],
  function(x) MergeSuffix(x, y3, delim = c("space", "dash")))

# Generate KWIC index
GNKWIC <- KWIC(GN, GNfields)

# Specify the exceptions as a vector
  "DARK", "E", "EARLY", "EC", "ERECT", "EXOTIC", "FLESH", "GROUNDNUT",
  "GUTHUKAI", "IMPROVED", "K", "KUTHUKADAL", "KUTHUKAI", "LARGE",
  "RED", "RUNNER", "S1", "SAM", "SMALL", "SPANISH", "TAN", "TYPE",
  "U", "VALENCIA", "VIRGINIA", "WHITE")

# Specify the synsets as a list
syn <- list(c("CHANDRA", "AH114"), c("TG1", "VIKRAM"))

# Fetch probable duplicate sets
GNdup <- ProbDup(kwic1 = GNKWIC, method = "a", excep = exep, fuzzy = TRUE,
  phonetic = TRUE, encoding = "primary",
  semantic = TRUE, syn = syn)

# Initial number of sets
GNdup

Method : a

KWIC1 fields : NationalID CollNo DonorID OtherID1 OtherID2

<table>
<thead>
<tr>
<th></th>
<th>No..of.Sets</th>
<th>No..of.Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>FuzzyDuplicates</td>
<td>378</td>
<td>745</td>
</tr>
<tr>
<td>PhoneticDuplicates</td>
<td>99</td>
<td>260</td>
</tr>
<tr>
<td>SemanticDuplicates</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Total</td>
<td>479</td>
<td>1010</td>
</tr>
<tr>
<td>(Distinct:762)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

# Get disjoint probable duplicate sets of each kind
disGNdup1 <- DisProbDup(GNdup, combine = NULL)

# # Number of sets after combining intersecting sets
disGNdup1

Method : a

KWIC1 fields : NationalID CollNo DonorID OtherID1 OtherID2

<table>
<thead>
<tr>
<th></th>
<th>No..of.Sets</th>
<th>No..of.Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>FuzzyDuplicates</td>
<td>181</td>
<td>745</td>
</tr>
<tr>
<td>PhoneticDuplicates</td>
<td>80</td>
<td>260</td>
</tr>
<tr>
<td>SemanticDuplicates</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Total</td>
<td>263</td>
<td>1010</td>
</tr>
<tr>
<td>(Distinct:762)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

# Get disjoint probable duplicate sets combining all the kinds of sets
disGNdup2 <- DisProbDup(GNdup, combine = c("F", "P", "S"))
An Introduction to PGRdup Package

Set Review, Modification and Validation

# Number of sets after combining intersecting sets
disGNdup2

Method: a

KWIC1 fields: NationalID CollNo DonorID OtherID1 OtherID2

<table>
<thead>
<tr>
<th></th>
<th>No.of.Sets</th>
<th>No.of.Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>DisjointDuplicates</td>
<td>167</td>
<td>762</td>
</tr>
<tr>
<td>Total</td>
<td>167</td>
<td>762 (Distinct: 762)</td>
</tr>
</tbody>
</table>

Once duplicate sets are retrieved they can be validated by manual clerical review by comparing with original PGR passport database(s) using the `ReviewProbDup` function. This function helps to retrieve PGR passport information associated with fuzzy, phonetic or semantic probable duplicate sets in an object of class `ProbDup` from the original databases(s) from which they were identified. The original information of accessions comprising a set, which have not been subjected to data standardization can be compared under manual clerical review for the validation of the set. By default only the fields(columns) which were used initially for creation of the KWIC indexes using the KWIC function are retrieved. Additional fields(columns) if necessary can be specified using the `extra.db1` and `extra.db2` arguments.

When any primary ID/key records in the fuzzy, phonetic or semantic duplicate sets are found to be missing from the original databases specified in `db1` and `db2`, then they are ignored and only the matching records are considered for retrieving the information with a warning.

This may be due to data standardization of the primary ID/key field using the function `DataClean` before creation of the KWIC index and subsequent identification of probable duplicate sets. In such a case, it is recommended to use an identical data standardization operation on the primary ID/key field of databases specified in `db1` and `db2` before running this function.

With R <= v3.0.2, due to copying of named objects by `list()`, `Invalid .internal.selfref detected and fixed...` warning can appear, which may be safely ignored.

The output data frame can be subjected to clerical review either after exporting into an external spreadsheet using `write.csv` function or by using the `edit` function.

The column DEL can be used to indicate whether a record has to be deleted from a set or not. Y indicates “Yes”, and the default N indicates “No”.

The column SPLIT similarly can be used to indicate whether a record in a set has to be branched into a new set. A set of identical integers in this column other than the default 0 can be used to indicate that they are to be removed and assembled into a new set.

# Load the original database and clean the Primary ID/key field

```r
GN1000 <- GN1000
GN1000$NationalID <- DataClean(GN1000$NationalID)
```

# Get the data frame for reviewing the duplicate sets identified

```r
RevGNdup <- ReviewProbDup(pdup = disGNdup1, db1 = GN1000,
extra.db1 = c("SourceCountry", "TransferYear"),
max.count = 30, insert.blanks = TRUE)
```

```r
head(RevGNdup)

<table>
<thead>
<tr>
<th>SET_NO</th>
<th>TYPE</th>
<th>K[a]</th>
<th>PRIM_ID</th>
<th>IDKW</th>
<th>DEL</th>
<th>SPLIT</th>
<th>COUNT</th>
<th>K1_NationalID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>[K1]</td>
<td>EC100277</td>
<td>[K1]EC100277:U44712</td>
<td>N</td>
<td>0</td>
<td>3</td>
<td>EC100277</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>[K1]</td>
<td>EC21118</td>
<td>[K1]EC21118:U44712</td>
<td>N</td>
<td>0</td>
<td>3</td>
<td>EC21118</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>[K1]</td>
<td>IC494796</td>
<td>[K1]IC494796:U44712</td>
<td>N</td>
<td>0</td>
<td>3</td>
<td>IC494796</td>
</tr>
<tr>
<td>4</td>
<td>NA</td>
<td>&lt;NA&gt;</td>
<td>&lt;NA&gt;</td>
<td>&lt;NA&gt;</td>
<td>&lt;NA&gt;</td>
<td>&lt;NA&gt;</td>
<td>&lt;NA&gt;</td>
<td>&lt;NA&gt;</td>
</tr>
</tbody>
</table>
```
An Introduction to PGRdup Package

Set Review, Modification and Validation

5 1 P [K1] EC100713 [K1]EC100713:STARR N 0 14 EC100713
6 1 P [K1] EC106985 [K1]EC106985:STARR N 0 14 EC106985

K1_CollNo K1_DonorID K1_OtherID1 K1_OtherID2
1 Shulamith/ NRCG-14555 ICG-4709 U4-47-12
2 U 4-47-12; EC 21118; UKA ICG3265 U44712 U K A
3 U-4-47-12 ICG-6890 U44712
4 <NA> <NA> <NA> <NA>
5 EC 100713; ICG5296 STARR
6 Starr ICG3479

K1X_SourceCountry K1X_TransferYear
1 Israel 2014
2 Australia 1989
3 Unknown 2010
4 <NA> NA
5 United States of America 2004
6 United States of America 2001

# Examine and review the duplicate sets using edit function
RevGNdup <- edit(RevGNdup)

# OR examine and review the duplicate sets after exporting them as a csv file
write.csv(file="Duplicate sets for review.csv", x=RevGNdup)

After clerical review, the data frame created using the function ReviewProbDup from an object of class ProbDup can be reconstituted back to the same object after the review using the function ReconstructProbDup.

The instructions for modifying the sets entered in the appropriate format in the columns DEL and SPLIT during clerical review are taken into account for reconstituting the probable duplicate sets. Any records with Y in column DEL are deleted and records with identical integers in the column SPLIT other than the default 0 are reassembled into a new set.

# The original set data
data(RevGNdup, SET_NO==13 & TYPE=="P", select= c(IDKW, DEL, SPLIT))

IDKW DEL SPLIT
111 [K1]EC38607:MANFREDI1 N 0
112 [K1]EC420966:MANFREDI N 0
113 [K1]EC42549:MANFREDI68 N 0
114 [K1]EC42550:MANFREDI N 0
116 [K1]EC573128:MANFREDI84 N 0
117 [K1]IC304523:CHAMPAGUE, [K1]IC304523:MANFREDI N 0

# Make dummy changes to the set for illustration
RevGNdup[113, 116, 6] <- "Y"
RevGNdup[111, 114, 7] <- 1

# The instruction for modification in columns DEL and SPLIT
data(RevGNdup, SET_NO==13 & TYPE=="P", select= c(IDKW, DEL, SPLIT))

IDKW DEL SPLIT
111 [K1]EC38607:MANFREDI1 N 1
112 [K1]EC420966:MANFREDI N 2
113 [K1]EC42549:MANFREDI68 Y 0
114 [K1]EC42550:MANFREDI N 1
116 [K1]EC573128:MANFREDI84 Y 0
An Introduction to PGRdup Package

Other Functions

# Reconstruct ProDup object
GNdup2 <- ReconstructProbDup(RevGNdup)
# Initial no. of sets
disGNdup1

Method : a

KWIC1 fields : NationalID CollNo DonorID OtherID1 OtherID2

<table>
<thead>
<tr>
<th>No.of.Sets</th>
<th>No.of.Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>FuzzyDuplicates</td>
<td>181</td>
</tr>
<tr>
<td>PhoneticDuplicates</td>
<td>80</td>
</tr>
<tr>
<td>SemanticDuplicates</td>
<td>2</td>
</tr>
<tr>
<td>Total</td>
<td>263</td>
</tr>
</tbody>
</table>

# No. of sets after modifications
GNdup2

Method : a

KWIC1 fields : NationalID CollNo DonorID OtherID1 OtherID2

<table>
<thead>
<tr>
<th>No.of.Sets</th>
<th>No.of.Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>FuzzyDuplicates</td>
<td>180</td>
</tr>
<tr>
<td>PhoneticDuplicates</td>
<td>81</td>
</tr>
<tr>
<td>SemanticDuplicates</td>
<td>2</td>
</tr>
<tr>
<td>Total</td>
<td>263</td>
</tr>
</tbody>
</table>

Other Functions

The ProbDup object is a list of data frames of different kinds of probable duplicate sets viz- FuzzyDuplicates, PhoneticDuplicates, SemanticDuplicates and DisjointDuplicates. Each row of the component data frame will have information of a set, the type of set, the set members as well as the keywords based on which the set was formed. This data can be reshaped into long form using the function ParseProbDup. This function which will transform a ProbDup object into a single data frame.

# Convert 'ProbDup' object to a long form data frame of sets
GNdupParsed <- ParseProbDup(GNdup)
head(GNdupParsed)

<table>
<thead>
<tr>
<th>SET_NO</th>
<th>TYPE</th>
<th>K</th>
<th>PRIM_ID</th>
<th>IDKW</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>F</td>
<td>[K1]IC494796</td>
<td>[K1]IC494796:U44712</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>NA</td>
<td>&lt;NA&gt;</td>
<td>&lt;NA&gt;</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>[K1]EC100280</td>
<td>[K1]EC100280:NC5</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

The prefix K* here indicates the KWIC index of origin. This is useful in ascertaining the database of origin of the accessions when method "b" or "c" was used to create the input ProbDup object.

Once the sets are reviewed and modified, the validated set data fields from the ProbDup object can be added to the original PGR passport database using the function AddProbDup. The associated data fields such as SET_NO, ID and IDKW are added based on the PRIM_ID field(column).
# Loading original database

GN2 <- GN1000

# Add the duplicates set data to the original database

GNwithdup <- AddProbDup(pdup = GNdup, db = GN2, addto = "I")

In case more than one KWIC index was used to generate the object of class ProbDup, the argument addto can be used to specify to which database the data fields are to be added. The default "I" indicates the database from which the first KWIC index was created and "II" indicates the database from which the second index was created.

The function SplitProbDup can be used to split an object of class ProbDup into two on the basis of set counts. This is useful for reviewing separately the sets with larger set counts.

# Load PGR passport database

GN <- GN1000

# Specify as a vector the database fields to be used

GNfields <- c("NationalID", "CollNo", "DonorID", "OtherID1", "OtherID2")

# Clean the data

GN[GNfields] <- lapply(GN[GNfields], function(x) DataClean(x))

y1 <- list(c("Gujarat", "Dwarf"), c("Castle", "Cary"), c("Small", "Japan"),
           c("Mani", "Blanco"), c("Uganda", "Erect"),
           c("Mota", "Company"))

y2 <- c("Dark", "Light", "Small", "Improved", "Punjab", "SAM")

y3 <- c("Local", "Bold", "Cary", "Mutant", "Runner", "Giant", "No.",
        "Bunch", "Peaunt")

GN[GNfields] <- lapply(GN[GNfields], function(x) MergeKW(x, y1, delim = c("space", "dash")))

GN[GNfields] <- lapply(GN[GNfields], function(x) MergePrefix(x, y2, delim = c("space", "dash")))

GN[GNfields] <- lapply(GN[GNfields], function(x) MergeSuffix(x, y3, delim = c("space", "dash")))

# Generate KWIC index

GNKWIC <- KWIC(GN, GNfields)

# Specify the exceptions as a vector

         "DARK", "E", "EARLY", "EC", "ERECT", "EXOTIC", "FLESH", "GROUNDNUT",
         "GUTHUKAI", "IMPROVED", "K", "KUTHUKADAL", "KUTHUKAI", "LARGE",
         "RED", "RUNNER", "S1", "SAM", "SMALL", "SPANISH", "TAN", "TYPE",
         "U", "VALENCIA", "VIRGINIA", "WHITE")

# Specify the synsets as a list

syn <- list(c("CHANDRA", "AH114"), c("TG1", "VIKRAM"))

# Fetch probable duplicate sets

GNdup <- ProbDup(kwic1 = GNKWIC, method = "a", excep = exep, fuzzy = TRUE,
                 phonetic = TRUE, encoding = "primary",
                 semantic = TRUE, syn = syn)
# Split the probable duplicate sets

```r
gNdupSplit <- SplitProbDup(GNdup, splitat = c(10, 10, 10))
gNdupSplit[[1]]
```

Method: `a`

KWIC1 fields: NationalID CollNo DonorID OtherID1 OtherID2

<table>
<thead>
<tr>
<th></th>
<th>No. of Sets</th>
<th>No. of Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>FuzzyDuplicates</td>
<td>338</td>
<td>744</td>
</tr>
<tr>
<td>PhoneticDuplicates</td>
<td>99</td>
<td>260</td>
</tr>
<tr>
<td>SemanticDuplicates</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Total</td>
<td>439</td>
<td>1009 (Distinct:762)</td>
</tr>
</tbody>
</table>

```
gNdupSplit[[3]]
```

Method: `a`

KWIC1 fields: NationalID CollNo DonorID OtherID1 OtherID2

<table>
<thead>
<tr>
<th></th>
<th>No. of Sets</th>
<th>No. of Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>FuzzyDuplicates</td>
<td>40</td>
<td>136</td>
</tr>
<tr>
<td>PhoneticDuplicates</td>
<td></td>
<td>136 (Distinct:136)</td>
</tr>
<tr>
<td>SemanticDuplicates</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Total</td>
<td>40</td>
<td>136 (Distinct:136)</td>
</tr>
</tbody>
</table>

Alternatively, two different `ProbDup` objects can be merged together using the function `MergeProbDup`.

```r
gNdupMerged <- MergeProbDup(gNdupSplit[[1]], gNdupSplit[[3]])
gNdupMerged
```

Method: `a`

KWIC1 fields: NationalID CollNo DonorID OtherID1 OtherID2

<table>
<thead>
<tr>
<th></th>
<th>No. of Sets</th>
<th>No. of Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>FuzzyDuplicates</td>
<td>378</td>
<td>745</td>
</tr>
<tr>
<td>PhoneticDuplicates</td>
<td>99</td>
<td>260</td>
</tr>
<tr>
<td>SemanticDuplicates</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Total</td>
<td>479</td>
<td>1010 (Distinct:762)</td>
</tr>
</tbody>
</table>

The summary of accessions according to a grouping factor field (column) in the original database(s) within the probable duplicate sets retrieved in a `ProbDup` object can be visualized by the `ViewProbDup` function. The resulting plot can be used to examine the extent of probable duplication within and between groups of accessions records.

# Load PGR passport databases

```r
gN1 <- GN1000[!grepl("^ICG", GN1000$DonorID), ]
gN1$DonorID <- NULL
gN2 <- GN1000[grepl("^ICG", GN1000$DonorID), ]
gN2 <- GN2[!grepl("^ICG", GN2$DonorID), ]
gN2$NationalID <- NULL
gN1$SourceCountry <- toupper(gN1$SourceCountry)
gN2$SourceCountry <- toupper(gN2$SourceCountry)
gN1$SourceCountry <- gsub("UNITED STATES OF AMERICA", "USA", gN1$SourceCountry)
gN2$SourceCountry <- gsub("UNITED STATES OF AMERICA", "USA", gN2$SourceCountry)
```
# Specify as a vector the database fields to be used
GN1fields <- c("NationalID", "CollNo", "OtherID1", "OtherID2")
GN2fields <- c("DonorID", "CollNo", "OtherID1", "OtherID2")

# Clean the data
GN1[GN1fields] <- lapply(GN1[GN1fields], function(x) DataClean(x))
GN2[GN2fields] <- lapply(GN2[GN2fields], function(x) DataClean(x))

# Remove duplicated DonorID records in GN2
GN2 <- GN2[!duplicated(GN2$DonorID),]

# Generate KWIC index
GN1KWIC <- KWIC(GN1, GN1fields)
GN2KWIC <- KWIC(GN2, GN2fields)

# Specify the exceptions as a vector

# Specify the synsets as a list
syn <- list(c("CHANDRA", "AH114"), c("TG1", "VIKRAM"))

# Probabilities of duplicated records in GN2
GNdupc <- ProbDup(kwic1 = GN1KWIC, kwic2 = GN2KWIC, method = "c",
                  excep = exep, fuzzy = TRUE, phonetic = TRUE,
                  encoding = "primary", semantic = TRUE, syn = syn)
Semantic matching

# Get the summary data.frames and Grob
GNdupcView <- ViewProbDup(GNdupc, GN1, GN2, "SourceCountry", "SourceCountry",
                          max.count = 30, select = c("INDIA", "USA"), order = "type",
                          main = "Groundnut Probable Duplicates")

# View the summary data.frames
GNdupcView[[1]]
GNdupcView[[2]]

# Plot the summary visualization
library(gridExtra)
grid.arrange(GNdupcView[[3]])

Fig. 5. Summary visualization of groundnut probable duplicate sets retrieved according to SourceCountry field.

The function KWCounts can be used to compute the keyword counts from PGR passport database fields(columns) which are considered for identification of probable duplicates. These keyword counts can give a rough indication of the completeness of the data in such fields (Fig. 3).

# Compute the keyword counts for the whole data
GNKWCounts <- KWCounts(GN, GNfields, exep)

# Compute the keyword counts for 'duplicated' records
GND <- ParseProbDup(disGNdup2, Inf, F)$PRIM_ID
GNDKWCouts <- KWCounts(GN[GN$NationalID %in% GND, ], GNfields, exep)

# Compute the keyword counts for 'unique' records
GNUKWCouts <- KWCounts(GN[!GN$NationalID %in% GND, ], GNfields, exep)

# Plot the counts as barplot
par(mfrow = c(3,1))

bp1 <- barplot(table(GNKWCouts$COUNT),
               xlab = "Word count", ylab = "Frequency",
               main = "A", col = "#1B9E77")
text(bp1, 0, table(GNKWCouts$COUNT), cex = 1, pos = 3)
legend("topright", paste("No. of records =",
                          nrow(GN)),
       bty = "n")

bp2 <- barplot(table(GNDKWCouts$COUNT),
               xlab = "Word count", ylab = "Frequency",
               main = "B", col = "#D95F02")
text(bp2, 0, table(GNDKWCouts$COUNT), cex = 1, pos = 3)
legend("topright", paste("No. of records =",
                          nrow(GN[GN$NationalID %in% GND, ])),
       bty = "n")

bp3 <- barplot(table(GNUKWCouts$COUNT),
               xlab = "Word count", ylab = "Frequency",
               main = "C", col = "#7570B3")
text(bp3, 0, table(GNUKWCouts$COUNT), cex = 1, pos = 3)
legend("topright", paste("No. of records =",
                          nrow(GN[!GN$NationalID %in% GND, ])),
       bty = "n")
Fig. 6. The keyword counts in the database fields considered for identification of probable duplicates for A. the entire GN1000 dataset, B. the probable duplicate records alone and C. the unique records alone.
Citing PGRdup

citation("PGRdup")

To cite the R package 'PGRdup' in publications use:


A BibTeX entry for LaTeX users is

```latex
@Manual{,
  title = {PGRdup: Discover Probable Duplicates in Plant Genetic Resources Collections},
  author = {J. Aravind and J. Radhamani and Kalyani Srinivasan and B. Ananda Subhash and Rishi Kumar},
  year = {2020},
  note = {R package version 0.2.3.6},
  note = {https://github.com/aravind-j/PGRdup},
  note = {https://cran.r-project.org/package=PGRdup},
}
```

This free and open-source software implements academic research by the authors and co-workers. If you use it, please support the project by citing the package.

Session Info

```r
sessionInfo()
```

R Under development (unstable) (2020-07-22 r78897)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 10 x64 (build 18362)

Matrix products: default

locale:
[1] LC_COLLATE=C
[3] LC_MONETARY=English_India.1252
[5] LC_TIME=English_India.1252

attached base packages:
[1] stats graphics grDevices utils datasets methods base

other attached packages:
[1] PGRdup_0.2.3.6 gridExtra_2.3 wordcloud_2.6 RColorBrewer_1.1-2
[5] diagram_1.6.4 shape_1.4.4

loaded via a namespace (and not attached):
[1] Rcpp_1.0.5 highr_0.8 compiler_4.1.0
[4] pillar_1.4.6 bitops_1.0-6 tools_4.1.0
[7] digest_0.6.25 evaluate_0.14 lifecycle_0.2.0
[10] tibble_3.0.3 gtable_0.3.0 pkgconfig_2.0.3
An Introduction to PGRdup Package

Session Info

[13] rlang_0.4.7  igraph_1.2.5  microbenchmark_1.4-7
[16] curl_4.3_1  yaml_2.2.1  parallel_4.1.0
[19] xfun_0.16_1  httr_1.4.2  stringr_1.4.0
[22] dplyr_1.0.0  knitr_1.29  generics_0.0.2
[25] vctrs_0.3.2  grid_4.1.0  tidyselect_1.1.0
[28] glue_1.4.1_1  data.table_1.13.0  R6_2.4.1
[31] XML_3.99-0.5_1  rmarkdown_2.3  farver_2.0.3
[34] ggplot2_3.3.2_1  purrr_0.3.4  magrittr_1.5
[37] scales_1.1.1_1  htmltools_0.5.0  ellipsis_0.3.1
[40] stringdist_0.9.6_1  colorspace_1.4-1  labeling_0.3
[43] stringi_1.4.6_1  RColorBrewer_1.1-2  munsell_0.5.0
[46] crayon_1.3.4_1

References


