R Interface to CoreArray Genomic Data Structure (GDS) files

Xiuwen Zheng*

Jan 5, 2015

Contents

1 Introduction

The package provides a high-level *R* interface to CoreArray Genomic Data Structure (GDS) data files, which are portable across platforms and include hierarchical structure to store multiple scalable array-oriented data sets with metadata information. It is suited for large-scale datasets, especially for data which are much larger than the available random-access memory. The package *gdsfmt* offers the efficient operations specifically designed for integers with less than 8 bits, since a single genetic/genomic variant, like single-nucleotide polymorphism (SNP), usually occupies fewer bits than a byte. Data compression and decompression are also supported with relatively efficient random access.

*zhengx@u.washington.edu

2 Installation of the package gdsfmt

To install the package *gdsfmt*, you need a current version (¿=2.14.0) of *R* (www.r-project.org). After installing *R* you can run the following commands from the *R* command shell to install the R package *gdsfmt*.

Install the development version from Github:

```
library("devtools")
install_github("zhengxwen/gdsfmt")
```

3 High-level R functions

3.1 Creating a GDS file and variable hierarchy

An empty GDS file can be created by `createfn.gds`:

```
library(gdsfmt)
gfile <- createfn.gds("test.gds")
```

Now, a file handle associated with 'test.gds' is saved in the R variable `gfile`.

The GDS file can contain a hierarchical structure to store multiple GDS variables (or GDS nodes) in the file, and various data types are allowed (see the document of `add.gdsn`) including integer, floating-point number and character.

```
add.gdsn(gfile, "int", val=1:10000)
add.gdsn(gfile, "double", val=seq(1, 1000, 0.4))
add.gdsn(gfile, "character", val=c("int", "double", "logical", "factor"))
add.gdsn(gfile, "logical", val=rep(c(TRUE, FALSE, NA), 50), visible=FALSE)
add.gdsn(gfile, "factor", val=as.factor(c(NA, "AA", "CC")), visible=FALSE)
add.gdsn(gfile, "bit2", val=sample(0:3, 1000, replace=TRUE), storage="bit2")


# list and data.frame
add.gdsn(gfile, "list", val=list(X=1:10, Y=seq(1, 10, 0.25)))
add.gdsn(gfile, "data.frame", val=data.frame(X=1:19, Y=seq(1, 10, 0.5)))
```

```
folder <- addfolder.gdsn(gfile, "folder")
add.gdsn(folder, "int", val=1:1000)
add.gdsn(folder, "double", val=seq(1, 100, 0.4), visible=FALSE)
```

Users can display the file content by typing `gfile` or `print(gfile)`:

```
gfile
```

```
## File: /tmp/RtmprEdTMo/Rbuildd6538aa4d12/gdsfmt/vignettes/test.gds
## +    [  ]
## |--+ int   { Int32 10000 }
## |--+ double   { Float64 2498 }
## |--+ character   { VStr8 4 }
## |--+ bit2   { Bit2 1000 }
## |--+ list   [ list ] *
## |  |--+ X   { Int32 10 }
## |  |--+ Y   { Float64 37 }
## |--+ data.frame   [ data.frame ] *
## |  |--+ X   { Int32 19 }
## |  |--+ Y   { Float64 19 }
## |--+ folder   [  ]
## |  |--+ int   { Int32 1000 }
```

`print(gfile, ...)` has an argument `all` to control the display of file content. By default, `all=FALSE`; if `all=TRUE`, to show all contents in the file including hidden variables or folders. The GDS variables `logical`, `factor` and `folder/double` are hidden.

```
print(gfile, all=TRUE)
```

```
## File: /tmp/RtmprEdTMo/Rbuildd6538aa4d12/gdsfmt/vignettes/test.gds
## +    [  ]
## |--+ int   { Int32 10000 }
## |--+ double   { Float64 2498 }
## |--+ character   { VStr8 4 }
## |--+ logical   { Int32,logical 150 } *
## |--+ factor   { Int32,factor 3 } *
## |--+ bit2   { Bit2 1000 }
## |--+ list   [ list ] *
## |  |--+ X   { Int32 10 }
## |  |--+ Y   { Float64 37 }
## |--+ data.frame   [ data.frame ] *
## |  |--+ X   { Int32 19 }
## |  |--+ Y   { Float64 19 }
## |--+ folder   [  ]
## |  |--+ int   { Int32 1000 }
## |  |--+ double   { Float64 248 } *
```

The asterisk indicates attributes attached to a GDS variable. The attributes can be used in the R environment to interpret the variable as *logical* , *factor*, *data.frame* or *list*.

`index.gdsn` can locate the GDS variable by a `path`:

```
index.gdsn(gfile, "int")
```

```
## + int   { Int32 10000 }
```

```
index.gdsn(gfile, "list/Y")
```

```
## + list/Y   { Float64 37 }
```

```
index.gdsn(gfile, "folder/int")
```

```
## + folder/int   { Int32 1000 }
```

```
# close the GDS file
closefn.gds(gfile)
```

3.2 Writing Data

Array-oriented data sets can be written to the GDS file. There are three possible ways to write data to a GDS variable.

```
gfile <- createfn.gds("test.gds")
```

3.2.1 R function `add.gdsn`

Users could pass an R variable to the function `add.gdsn` directly. `read.gdsn` can read data from the GDS variable.

```
(n <- add.gdsn(gfile, "I1", val=matrix(1:15, nrow=3)))
```

```
## + I1   { Int32 3x5 }
```

```
read.gdsn(n)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    4    7   10   13
## [2,]    2    5    8   11   14
## [3,]    3    6    9   12   15
```

3.2.2 R function `write.gdsn`

Users can specify the arguments `start` and `count` to write a subset of data. -1 in `count` means the size of that dimension, and the corresponding element in `start` should be 1. The values in `start` and `cound` should be in the dimension range.

```
write.gdsn(n, rep(0,5), start=c(2,1), count=c(1,-1))
read.gdsn(n)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    4    7   10   13
## [2,]    0    0    0    0    0
```

```
## [3,]    3    6    9   12   15
```

3.2.3 R function append.gdsn

Users can append new data to an existing GDS variable.

```
append.gdsn(n, 16:24)
read.gdsn(n)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    1    4    7   10   13   16   19   22
## [2,]    0    0    0    0    0   17   20   23
## [3,]    3    6    9   12   15   18   21   24
```

3.2.4 Create a large-scale data set

When the size of dataset is larger than the system memory, users can not add a GDS variable via add.gdsn directly. If the dimension is pre-defined, users can specify the dimension size in add.gdsn to allocate data space. Then call write.gdsn to write a small subset of data space.

```
(n2 <- add.gdsn(gfile, "I2", storage="int", valdim=c(100, 2000)))
```

```
## + I2   { Int32 100x2000 }
```

```
for (i in 1:2000)
{
    write.gdsn(n2, seq.int(100*(i-1)+1, length.out=100),
        start=c(1,i), count=c(-1,1))
}
```

Or call append.gdsn to append new data when the initial size is ZERO. If a compression algorithm is specified in add.gdsn (e.g., compress="ZIP"), users should call append.gdsn instead of write.gdsn, since data has to be compressed sequentially.

```
(n3 <- add.gdsn(gfile, "I3", storage="int", valdim=c(100, 0), compress="ZIP"))
```

```
## + I3   { Int32 100x0 ZIP }
```

```
for (i in 1:2000)
{
    append.gdsn(n3, seq.int(100*(i-1)+1, length.out=100))
}

n3
```

```
## + I3   { Int32 100x2000 ZIP(32.29%) }
```

```
# close the GDS file
closefn.gds(gfile)
```

3.3 Reading Data

`read.gdsn` can load all data to an R variable in memory.

```
gfile <- createfn.gds("test.gds")
(n <- add.gdsn(gfile, "I1", val=matrix(1:20, nrow=4)))
```

```
## + I1   { Int32 4x5 }
```

```
read.gdsn(n)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    5    9   13   17
## [2,]    2    6   10   14   18
## [3,]    3    7   11   15   19
## [4,]    4    8   12   16   20
```

3.3.1 Subset reading `read.gdsn` and `readex.gdsn`

A subset of data can be specified via the arguments `start` and `count` in the R function `read.gdsn`. Or specify a list of logical vectors in `readex.gdsn`.

```
# read a subset
read.gdsn(n, start=c(2, 2), count=c(2, 3))
```

```
##      [,1] [,2] [,3]
## [1,]    6   10   14
## [2,]    7   11   15
```

```
# read a subset
readex.gdsn(n, list(c(FALSE,TRUE,TRUE,FALSE), c(TRUE,FALSE,TRUE,FALSE,TRUE)))
```

```
##      [,1] [,2] [,3]
## [1,]    2   10   18
## [2,]    3   11   19
```

3.3.2 Apply a user-defined function marginally

A user-defined function can be applied marginally to a GDS variable via `apply.gdsn`. `margin=1` indicates applying the function row by row, and `margin=2` for applying the function column by column.

```
apply.gdsn(n, margin=1, FUN=print, as.is="none")
```

```
## [1]  1  5  9 13 17
## [1]  2  6 10 14 18
## [1]  3  7 11 15 19
## [1]  4  8 12 16 20
```

```
apply.gdsn(n, margin=2, FUN=print, as.is="none")
```

```
## [1] 1 2 3 4
## [1] 5 6 7 8
## [1]  9 10 11 12
## [1] 13 14 15 16
## [1] 17 18 19 20
```

3.3.3 Transpose a matrix

`apply.gdsn` allows that the data returned from the user-defined function `FUN` is directly written to a target GDS node `target.node`, when `as.is="gdsnode"` and `target.node` are both given. Little c in R is a generic function which combines its arguments, and it passes all data to the target GDS node in the following code:

```
n.t <- add.gdsn(gfile, "transpose", storage="int", valdim=c(5,0))

# apply the function over rows of matrix
apply.gdsn(n, margin=1, FUN=c, as.is="gdsnode", target.node=n.t)

# matrix transpose
read.gdsn(n.t)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
## [4,]   13   14   15   16
## [5,]   17   18   19   20
```

```
# close the GDS file
closefn.gds(gfile)
```

4 Session Info

```
toLatex(sessionInfo())
```

- R version 3.1.2 (2014-10-31), `x86_64-unknown-linux-gnu`

- Locale: LC_CTYPE=en_US.UTF-8, LC_NUMERIC=C, LC_TIME=en_US.UTF-8, LC_COLLATE=C, LC_MONETARY=en_US.UTF-8, LC_MESSAGES=en_US.UTF-8, LC_PAPER=en_US.UTF-8, LC_NAME=C, LC_ADDRESS=C, LC_TELEPHONE=C, LC_MEASUREMENT=en_US.UTF-8, LC_IDENTIFICATION=C
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: gdsfmt 1.2.2
- Loaded via a namespace (and not attached): BiocStyle 1.4.1, evaluate 0.5.5, formatR 1.0, highr 0.4, knitr 1.8, stringr 0.6.2, tools 3.1.2