ChemmineR-V2: Analysis of Small Molecule and Screening Data

Yiqun Cao, Tyler Backman, Yan Wang, Thomas Girke
Email contact: thomas.girke@ucr.edu

August 3, 2012

1 Introduction

ChemmineR is a cheminformatics package for analyzing drug-like small molecule and screening data in R. Its new version ChemmineR-V2 contains functions for processing SDFs (structure data files), molecule depictions, structural similarity searching, clustering/diversity analyses of compound libraries with a wide spectrum of algorithms and utilities for managing complex data sets from high-throughput compound bio-assays.

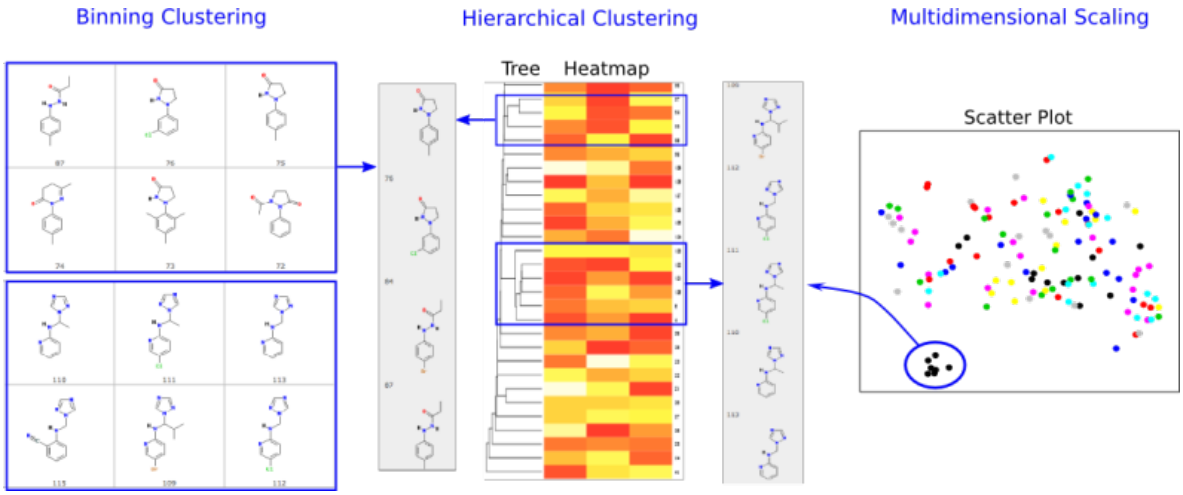


Figure 1: Selected functionalities provided by the *ChemmineR* package.

In addition, *ChemmineR* offers visualization functions for compound clustering results and chemical structures. The integration of chemoinformatic tools with the R programming environment has many advantages, such as easy access to a wide spectrum of statistical methods, machine learning algorithms and graphic utilities. The first version of this package was published in Cao et al. (2008). Since then many additional utilities have been added to the package and many more are under development for future releases (Backman et al., 2011).

2 Getting Started

2.1 Installation

The R software for running ChemmineR can be downloaded from CRAN (`http://cran.at.r-project.org/`). The ChemmineR package can be installed from R using the `bioLite` install command.

```
> source("http://bioconductor.org/biocLite.R") # Sources the biocLite.R installation script.
> biocLite("ChemmineR") # Installs the package.
```

2.2 Loading the Package and Documentation

```
> library("ChemmineR") # Loads the package
```

```
> library(help="ChemmineR") # Lists all functions and classes
> vignette("ChemmineR") # Opens this PDF manual from R
```

2.3 Five Minute Tutorial

The following code gives an overview of the most important functionalities provided by *ChemmineR*. Copy and paste of the commands into the R console will demonstrate their utilities.

Create Instances of *SDFset* class:

```
> data(sdfsample)
> sdfset <- sdfsample
> sdfset # Returns summary of SDFset

An instance of "SDFset" with 100 molecules

> sdfset[1:4] # Subsetting of object

An instance of "SDFset" with 4 molecules

> sdfset[[1]] # Returns summarized content of one SDF

An instance of "SDF"

<<header>>
                            Molecule_Name
                                 "650001"
                                   Source
                  "  -OEChem-07071010512D"
                                  Comment
                                       ""
                                Counts_Line
" 61 64  0     0  0  0  0  0  0999 V2000"

<<atomblock>>
```

```
          C1      C2  C3  C5  C6  C7  C8  C9 C10 C11 C12 C13 C14 C15 C16
O_1   7.0468  0.0839   0   0   0   0   0   0   0   0   0   0   0   0   0
O_2  12.2708  1.0492   0   0   0   0   0   0   0   0   0   0   0   0   0
...      ...     ... ... ... ... ... ... ... ... ... ... ... ... ... ...
H_60  1.8411 -1.5985   0   0   0   0   0   0   0   0   0   0   0   0   0
H_61  2.6597 -1.2843   0   0   0   0   0   0   0   0   0   0   0   0   0

<<bondblock>>
     C1  C2  C3  C4  C5  C6  C7
1     1  16   2   0   0   0   0
2     2  23   1   0   0   0   0
... ... ... ... ... ... ... ...
63   33  60   1   0   0   0   0
64   33  61   1   0   0   0   0

<<datablock>> (33 data items)
          PUBCHEM_COMPOUND_CID PUBCHEM_COMPOUND_CANONICALIZED
                      "650001"                            "1"
     PUBCHEM_CACTVS_COMPLEXITY  PUBCHEM_CACTVS_HBOND_ACCEPTOR
                         "700"                            "7"

                         "..."
```

```
> view(sdfset[1:4]) # Returns summarized content of many SDFs, not printed here
> as(sdfset[1:4], "list") # Returns complete content of many SDFs, not printed here
```

An *SDFset* is created during the import of an SD file:

```
> sdfset <- read.SDFset("http://faculty.ucr.edu/~tgirke/Documents/
+                       R_BioCond/Samples/sdfsample.sdf")
```

Miscellaneous accessor methods for *SDFset* container:

```
> header(sdfset[1:4]) # Not printed here

> header(sdfset[[1]])

                          Molecule_Name
                               "650001"
                                 Source
                "  -OEChem-07071010512D"
                                Comment
                                     ""
                            Counts_Line
" 61 64  0     0  0  0  0  0  0999 V2000"

> atomblock(sdfset[1:4]) # Not printed here

> atomblock(sdfset[[1]])[1:4,]
```

```
         C1     C2 C3 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16
O_1  7.0468 0.0839  0  0  0  0  0  0   0   0   0   0   0   0   0
O_2 12.2708 1.0492  0  0  0  0  0  0   0   0   0   0   0   0   0
O_3 12.2708 3.1186  0  0  0  0  0  0   0   0   0   0   0   0   0
O_4  7.9128 2.5839  0  0  0  0  0  0   0   0   0   0   0   0   0
```

```
> bondblock(sdfset[1:4]) # Not printed here
```

```
> bondblock(sdfset[[1]])[1:4,]
```

```
  C1 C2 C3 C4 C5 C6 C7
1  1 16  2  0  0  0  0
2  2 23  1  0  0  0  0
3  2 27  1  0  0  0  0
4  3 25  1  0  0  0  0
```

```
> datablock(sdfset[1:4]) # Not printed here
```

```
> datablock(sdfset[[1]])[1:4]
```

```
          PUBCHEM_COMPOUND_CID PUBCHEM_COMPOUND_CANONICALIZED
                      "650001"                             "1"
     PUBCHEM_CACTVS_COMPLEXITY  PUBCHEM_CACTVS_HBOND_ACCEPTOR
                         "700"                             "7"
```

Assigning compound IDs and keeping them unique:

```
> cid(sdfset)[1:4] # Returns IDs from SDFset object
```

```
[1] "CMP1" "CMP2" "CMP3" "CMP4"
```

```
> sdfid(sdfset)[1:4] # Returns IDs from SD file header block
```

```
[1] "650001" "650002" "650003" "650004"
```

```
> unique_ids <- makeUnique(sdfid(sdfset))
```

```
[1] "No duplicates detected!"
```

```
> cid(sdfset) <- unique_ids
```

Converting the data blocks in an *SDFset* to a matrix:

```
> blockmatrix <- datablock2ma(datablocklist=datablock(sdfset))
>    # Converts data block to matrix
> numchar <- splitNumChar(blockmatrix=blockmatrix)
>    # Splits to numeric and character matrix
> numchar[[1]][1:2,1:2] # Slice of numeric matrix
```

```
       PUBCHEM_COMPOUND_CID PUBCHEM_COMPOUND_CANONICALIZED
650001               650001                               1
650002               650002                               1

> numchar[[2]][1:2,10:11] # Slice of character matrix

       PUBCHEM_MOLECULAR_FORMULA
650001 "C23H28N4O6"
650002 "C18H23N5O3"
       PUBCHEM_OPENEYE_CAN_SMILES
650001 "CC1=CC(=NO1)NC(=O)CCC(=O)N(CC(=O)NC2CCCC2)C3=CC4=C(C=C3)OCCO4"
650002 "CN1C2=C(C(=O)NC1=O)N(C(=N2)NCCCO)CCCC3=CC=CC=C3"
```

Compute atom frequency matrix, molecular weight and formula:

```
> propma <- data.frame(MF=MF(sdfset), MW=MW(sdfset), atomcountMA(sdfset))
> propma[1:4, ]

                MF       MW  C  H N O S F Cl
650001  C23H28N4O6 456.4916 23 28 4 6 0 0  0
650002  C18H23N5O3 357.4069 18 23 5 3 0 0  0
650003 C18H18N4O3S 370.4255 18 18 4 3 1 0  0
650004 C21H27N5O5S 461.5346 21 27 5 5 1 0  0
```

Assign matrix data to data block:

```
> datablock(sdfset) <- propma
> datablock(sdfset[1])

$`650001`
          MF           MW            C            H            N            O
"C23H28N4O6"   "456.4916"         "23"         "28"          "4"          "6"
           S            F           Cl
         "0"          "0"          "0"
```

String searching in *SDFset* ():

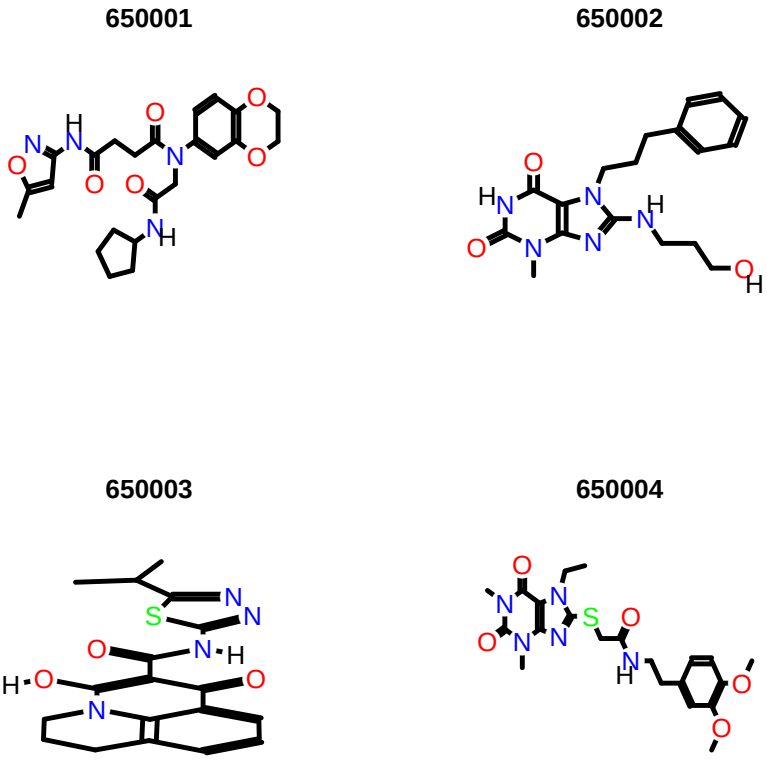
```
> grepSDFset("650001", sdfset, field="datablock", mode="subset")
>    # Returns summary view of matches. Not printed here.
> .

> grepSDFset("650001", sdfset, field="datablock", mode="index")

1 1 1 1 1 1 1 1 1
1 2 3 4 5 6 7 8 9
```

Export SDFset to SD file:

```
> write.SDF(sdfset[1:4], file="sub.sdf", sig=TRUE)
```

Plot molecule structure of one or many SDFs:

```
> plot(sdfset[1:4], print=FALSE) # Plots structures to R graphics device
```




```
> sdf.visualize(sdfset[1:4]) # Compound viewing in web browser
```

Structure similarity searching and clustering:

```
> apset <- sdf2ap(sdfset)
>    # Generate atom pair descriptor database for searching
> .

> data(apset)
>    # Load sample apset data provided by library.
> cmp.search(apset, apset[1], type=3, cutoff = 0.3, quiet=TRUE)

  index    cid    scores
1     1 650001 1.0000000
2    96 650102 0.3516643
3    67 650072 0.3117569
4    88 650094 0.3094629
5    15 650015 0.3010753
```

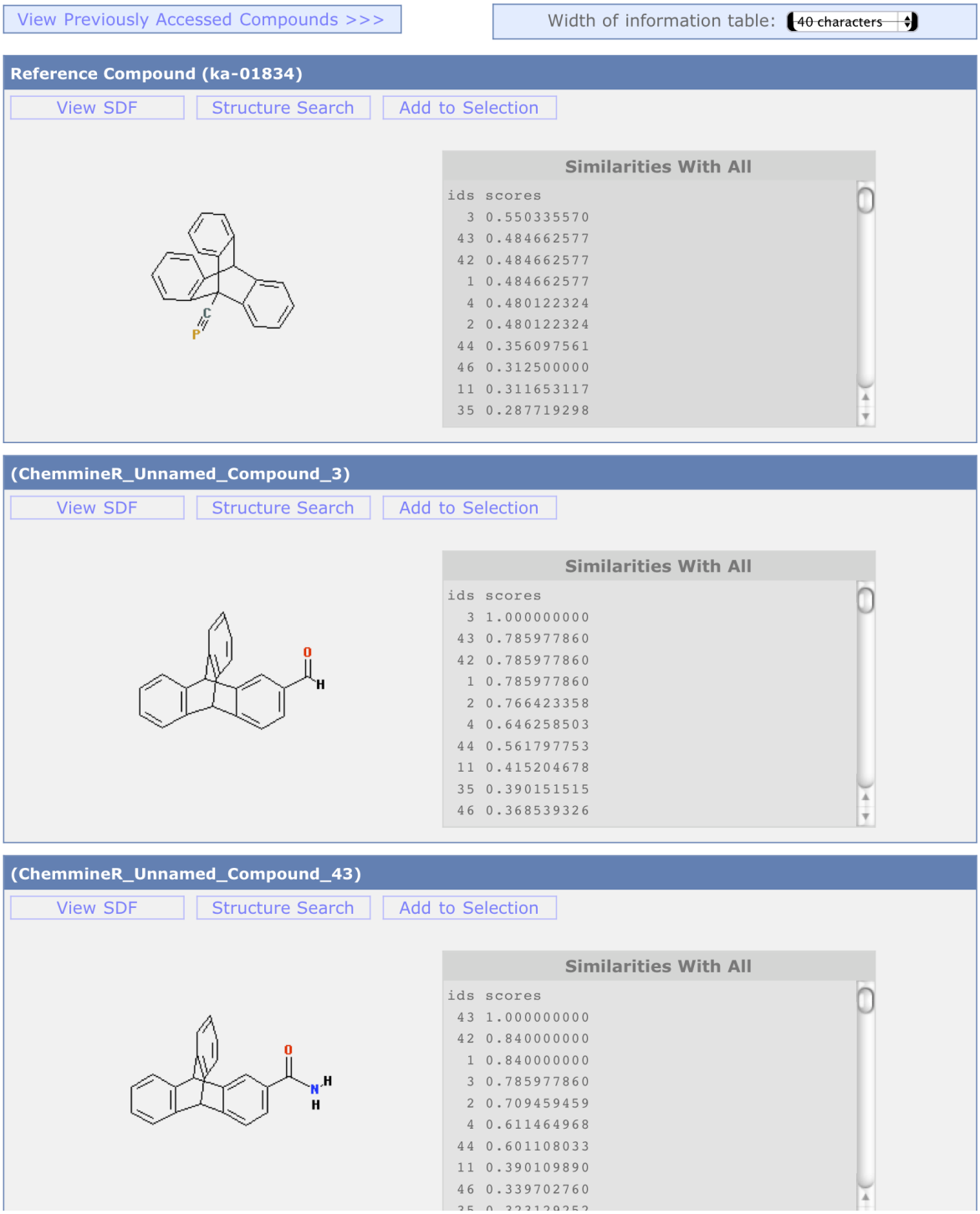


Figure 2: Visualization webpage created by calling `sdf.visualize`.

```
>    # Search apset database with single compound.
> cmp.cluster(db=apset, cutoff = c(0.65, 0.5), quiet=TRUE)[1:4,]

sorting result...
      ids CLSZ_0.65 CLID_0.65 CLSZ_0.5 CLID_0.5
48 650049         2        48        2       48
49 650050         2        48        2       48
54 650059         2        54        2       54
55 650060         2        54        2       54

>    # Binning clustering using variable similarity cutoffs.
```

3 Overview of Classes and Functions

The following list gives an overview of the most important S4 classes, methods and functions available in the ChemmineR package. The help documents of the package provide much more detailed information on each utility. The standard R help documents for these utilities can be accessed with this syntax: `?function_name` (*e.g.* `?cid`) and `?class_name-class` (*e.g.* `?"SDFset-class"`).

3.1 Molecular Structure Data

Classes

- *SDFstr*: intermediate string class to facilitate SD file import; not important for end user
- *SDF*: container for single molecule imported from an SD file
- *SDFset*: container for many SDF objects; most important structure container for end user

Functions/Methods

- Accessor methods for *SDF/SDFset*
  - Object slots: `cid, header, atomblock, bondblock, datablock (sdfid, datablocktag)`
  - Summary of *SDFset*: `view`
  - Matrix conversion of data block: `datablock2ma, splitNumChar`
  - String search in SDFset: `grepSDFset`
- Coerce one class to another
  - Standard syntax `as(..., "...")` works in most cases. For details see R help with *?"SDFset-class"*.
- Utilities
  - Atom frequencies: `atomcountMA, atomcount`

  - Molecular weight: `MW`
  - Molecular formula: `MF`

- Compound structure depictions
  - R graphics device: `plot, plotStruc`
  - Online: `cmp.visualize`

3.2 Structure Descriptor Data

Classes

- *AP*: container for atom pair descriptors of a single molecule
- *APset*: container for many AP objects; most important structure descriptor container for end user

Functions/Methods

- Create *AP/APset* instances
  - From *SDFset*: `sdf2ap`
  - From SD file: `cmp.parse`
  - Summary of *AP/APset*: `view, db.explain`
- Accessor methods for AP/APset
  - Object slots: `ap, cid`
- Coerce one class to another
  - Standard syntax `as(..., "...")` works in most cases. For details see R help with *?"APset-class"*.
- Structure Similarity comparisons and Searching
  - Compute pairwise similarities : `cmp.similarity`
  - Search APset database: `cmp.search`
  - Compute pairwise similarities : `cmp.similarity`
- AP-based Structure Similarity Clustering
  - Single-linkage binning clustering: `cmp.cluster`
  - Visualize clustering result with MDS: `cluster.visualize`
  - Size distribution of clusters: `cluster.sizestat`

4 Importing Compounds

The following code gives an overview of the most important import/export functionalities provided by *ChemmineR*. The example creates an instance of the *SDFset* class using as sample data set the first 100 compounds from this PubChem SD file (SDF): Compound_00650001_00675000.sdf.gz (`ftp://ftp.ncbi.nih.gov/pubchem/Compound/CURRENT-Full/SDF/`).

SDFs can be imported with the `read.SDFset` function:

```
> sdfset <- read.SDFset("http://faculty.ucr.edu/~tgirke/Documents/
+                        R_BioCond/Samples/sdfsample.sdf")

> data(sdfsample) # Loads the same SDFset provided by the library
> sdfset <- sdfsample
> valid <- validSDF(sdfset) # Identifies invalid SDFs in SDFset objects
> sdfset <- sdfset[valid] # Removes invalid SDFs, if there are any
```

Import SD file into *SDFstr* container:

```
> sdfstr <- read.SDFstr("http://faculty.ucr.edu/~tgirke/Documents/
+                        R_BioCond/Samples/sdfsample.sdf")
```

Create *SDFset* from *SDFstr* class:

```
> sdfstr <- as(sdfset, "SDFstr")
> sdfstr

An instance of "SDFstr" with 100 molecules

> as(sdfstr, "SDFset")

An instance of "SDFset" with 100 molecules
```

5 Export of Compounds

Write objects of classes *SDFset/SDFstr/SDF* to SD file:

```
> write.SDF(sdfset[1:4], file="sub.sdf")
```

Writing customized *SDFset* to file containing *ChemmineR* signature, IDs from *SDFset* and no data block:

```
> write.SDF(sdfset[1:4], file="sub.sdf", sig=TRUE, cid=TRUE, db=NULL)
```

Example for injecting a custom matrix/data frame into the data block of an *SDFset* and then writing it to an SD file:

```
> props <- data.frame(MF=MF(sdfset), MW=MW(sdfset), atomcountMA(sdfset))
> datablock(sdfset) <- props
> write.SDF(sdfset[1:4], file="sub.sdf", sig=TRUE, cid=TRUE)
```

Indirect export via *SDFstr* object:

```
> sdf2str(sdf=sdfset[[1]], sig=TRUE, cid=TRUE)
>    # Uses default components
> sdf2str(sdf=sdfset[[1]], head=letters[1:4], db=NULL)
>    # Uses custom components for header and data block
```

Write *SDF*, *SDFset* or *SDFstr* classes to file:

```
> write.SDF(sdfset[1:4], file="sub.sdf", sig=TRUE, cid=TRUE, db=NULL)
> write.SDF(sdfstr[1:4], file="sub.sdf")
> cat(unlist(as(sdfstr[1:4], "list")), file="sub.sdf", sep="\n")
```

6 Working with SDF/SDFset Classes

Several methods are available to return the different data components of *SDF/SDFset* containers in batches. The following examples list the most important ones. To save space their content is not printed in the manual.

```
> view(sdfset[1:4]) # Summary view of several molecules
> length(sdfset) # Returns number of molecules
> sdfset[[1]] # Returns single molecule from SDFset as SDF object
> sdfset[[1]][[2]] # Returns atom block from first compound as matrix
> sdfset[[1]][[2]][1:4,]
> c(sdfset[1:4], sdfset[5:8]) # Concatenation of several SDFsets
```

The `grepSDFset` function allows string matching/searching on the different data components in *SDFset*. By default the function returns a SDF summary of the matching entries. Alternatively, an index of the matches can be returned with the setting *mode="index"*.

```
> grepSDFset("650001", sdfset, field="datablock", mode="subset")
>    # To return index, set mode="index")
> .
```

Utilities to maintain unique compound IDs:

```
> sdfid(sdfset[1:4])
>    # Retrieves CMP IDs from Molecule Name field in header block.
> cid(sdfset[1:4])
>    # Retrieves CMP IDs from ID slot in SDFset.
> unique_ids <- makeUnique(sdfid(sdfset))
>    # Creates unique IDs by appending a counter to duplicates.
> cid(sdfset) <- unique_ids # Assigns uniquified IDs to ID slot
```

Subsetting by character, index and logical vectors:

```
> view(sdfset[c("650001", "650012")])
> view(sdfset[4:1])
> mylog <- cid(sdfset) %in% c("650001", "650012")
> view(sdfset[mylog])
```

Accessing *SDF/SDFset* components: header, atom, bond and data blocks:

```
> atomblock(sdf); sdf[[2]]; sdf[["atomblock"]]
>    # All three methods return the same component
> header(sdfset[1:4])
> atomblock(sdfset[1:4])
> bondblock(sdfset[1:4])
> datablock(sdfset[1:4])
> header(sdfset[[1]])
> atomblock(sdfset[[1]])
> bondblock(sdfset[[1]])
> datablock(sdfset[[1]])
```

Replacement Methods:

```
> sdfset[[1]][[2]][1,1] <- 999
> atomblock(sdfset)[1] <- atomblock(sdfset)[2]
> datablock(sdfset)[1] <- datablock(sdfset)[2]
```

Assign matrix data to data block:

```
> datablock(sdfset) <- as.matrix(iris[1:100,])
> view(sdfset[1:4])
```

Class coercions from *SDFstr* to *list*, *SDF* and *SDFset*:

```
> as(sdfstr[1:2], "list")
> as(sdfstr[[1]], "SDF")
> as(sdfstr[1:2], "SDFset")
```

Class coercions from *SDF* to *SDFstr*, *SDFset*, list with SDF sub-components:

```
> sdfcomplist <- as(sdf, "list")
> sdfcomplist <- as(sdfset[1:4], "list"); as(sdfcomplist[[1]], "SDF")
> sdflist <- as(sdfset[1:4], "SDF"); as(sdflist, "SDFset")
> as(sdfset[[1]], "SDFstr")
> as(sdfset[[1]], "SDFset")
```

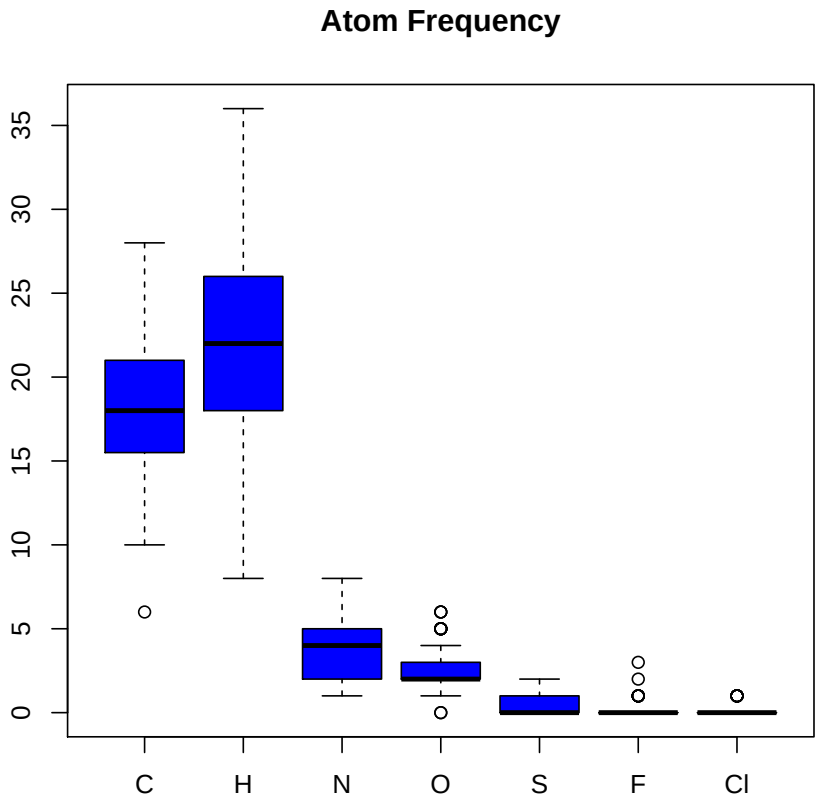
Class coercions from *SDFset* to lists with components consisting of SDF or sub-components:

```
> as(sdfset[1:4], "SDF")
> as(sdfset[1:4], "list")
> as(sdfset[1:4], "SDFstr")
```

7 Molecular Property Functions (Physicochemical Descriptors)

Several methods and functions are available to compute basic compound descriptors, such as molecular formula (MF), molecular weight (MW), and frequencies of atoms and functional groups. In many of these functions, it is important to set `addH=TRUE` in order to include/add hydrogens that are often not specified in an SD file.

```
> propma <- atomcountMA(sdfset, addH=FALSE)
> boxplot(propma, col="blue", main="Atom Frequency")
```




```
> boxplot(rowSums(propma), main="All Atom Frequency")
```

Data frame provided by library containing atom names, atom symbols, standard atomic weights, group and period numbers:

```
> data(atomprop)
> atomprop[1:4,]
```

```
  Number      Name Symbol Atomic_weight Group Period
1      1  hydrogen      H      1.007940     1      1
2      2    helium     He      4.002602    18      1
3      3   lithium     Li      6.941000     1      2
4      4 beryllium     Be      9.012182     2      2
```

Compute MW and formula:

```
> MW(sdfset[1:4], addH=FALSE)
```

```
    CMP1     CMP2     CMP3     CMP4
456.4916 357.4069 370.4255 461.5346

> MF(sdfset[1:4], addH=FALSE)

         CMP1           CMP2           CMP3           CMP4
 "C23H28N4O6"  "C18H23N5O3" "C18H18N4O3S" "C21H27N5O5S"
```

Enumerate functional groups:

```
> groups(sdfset[1:4], groups="fctgroup", type="countMA")

     RNH2 R2NH R3N ROPO3 ROH RCHO RCOR RCOOH RCOOR ROR
CMP1    0    2   1     0   0    0    0     0     0   2
CMP2    0    2   2     0   1    0    0     0     0   0
CMP3    0    1   1     0   1    0    1     0     0   0
CMP4    0    1   3     0   0    0    0     0     0   2
```

Combine MW, MF, charges, atom counts, functional group counts and ring counts in one data frame:

```
> propma <- data.frame(MF=MF(sdfset, addH=FALSE), MW=MW(sdfset, addH=FALSE),
+                      Ncharges=sapply(bonds(sdfset, type="charge"), length),
+                      atomcountMA(sdfset, addH=FALSE), groups(sdfset,
+                      type="countMA"), rings(sdfset, upper=6, type="count",
+                      arom=TRUE))
> propma[1:4,]

              MF       MW Ncharges  C  H N O S F Cl RNH2 R2NH R3N ROPO3 ROH
CMP1  C23H28N4O6 456.4916        0 23 28 4 6 0 0  0    0    2   1     0   0
CMP2  C18H23N5O3 357.4069        0 18 23 5 3 0 0  0    0    2   2     0   1
CMP3 C18H18N4O3S 370.4255        0 18 18 4 3 1 0  0    0    1   1     0   1
CMP4 C21H27N5O5S 461.5346        0 21 27 5 5 1 0  0    0    1   3     0   0
     RCHO RCOR RCOOH RCOOR ROR RINGS AROMATIC
CMP1    0    0     0     0   2     4        2
CMP2    0    0     0     0   0     3        3
CMP3    0    1     0     0   0     4        2
CMP4    0    0     0     0   2     3        3
```

The following shows an example for assigning the values stored in a matrix (*e.g.* property descriptors) to the data block components in an *SDFset*. Each matrix row will be assigned to the corresponding slot position in the *SDFset*.

```
> datablock(sdfset) <- propma # Works with all SDF components
> datablock(sdfset)[1:4]
> test <- apply(propma[1:4,], 1, function(x) data.frame(col=colnames(propma), value=x))
> sdf.visualize(sdfset[1:4], extra = test)
```

The data blocks in SDFs contain often important annotation information about compounds. The `datablock2ma` function returns this information as matrix for all compounds stored in an *SDFset* container. The `splitNumChar` function can then be used to organize all numeric columns in a *numeric matrix* and the character columns in a *character matrix* as components of a *list* object.

```
> datablocktag(sdfset, tag="PUBCHEM_NIST_INCHI")
> datablocktag(sdfset, tag="PUBCHEM_OPENEYE_CAN_SMILES")
```

Convert entire data block to matrix:

```
> blockmatrix <- datablock2ma(datablocklist=datablock(sdfset))
>    # Converts data block to matrix
> numchar <- splitNumChar(blockmatrix=blockmatrix)
>    # Splits matrix to numeric matrix and character matrix
> numchar[[1]][1:4,]; numchar[[2]][1:4,]
>    # Splits matrix to numeric matrix and character matrix
> .
```

8 Bond Matrices

Bond matrices provide an efficient data structure for many basic computations on small molecules. The function `conMA` creates this data structure from *SDF* and *SDFset* objects. The resulting bond matrix contains the atom labels in the row/column titles and the bond types in the data part. The labels are defined as follows: 0 is no connection, 1 is a single bond, 2 is a double bond and 3 is a triple bond.

```
> conMA(sdfset[1:2], exclude=c("H"))
>    # Create bond matrix for first two molecules in sdfset
> conMA(sdfset[[1]], exclude=c("H"))
>    # Return bond matrix for first molecule
> plot(sdfset[1], atomnum = TRUE, noHbonds=FALSE , no_print_atoms = "", atomcex=0.8)
>    # Plot its structure with atom numbering
> rowSums(conMA(sdfset[[1]], exclude=c("H")))
>    # Return number of non-H bonds for each atom
> .
```

9 Charges and Missing Hydrogens

The function `bonds` returns information about the number of bonds, charges and missing hydrogens in *SDF* and *SDFset* objects. It is used by many other functions (*e.g.* `MW`, `MF`, `atomcount`, `atomcuntMA` and `plot`) to correct for missing hydrogens that are often not specified in SD files.

```
> bonds(sdfset[[1]], type="bonds")[1:4,]
```

```
  atom Nbondcount Nbondrule charge
1    O          2         2      0
```

```
2    0          2         2      0
3    0          2         2      0
4    0          2         2      0
```

```
> bonds(sdfset[1:2], type="charge")
```

```
$CMP1
NULL

$CMP2
NULL
```

```
> bonds(sdfset[1:2], type="addNH")
```

```
CMP1 CMP2
   0    0
```

10 Ring Perception and Aromaticity Assignment

The function `rings` identifies all possible rings in one or many molecules (here *sdfset[1]*) using the exhaustive ring perception algorithm from Hanser et al. (1996). In addition, the function can return all smallest possible rings as well as aromaticity information.

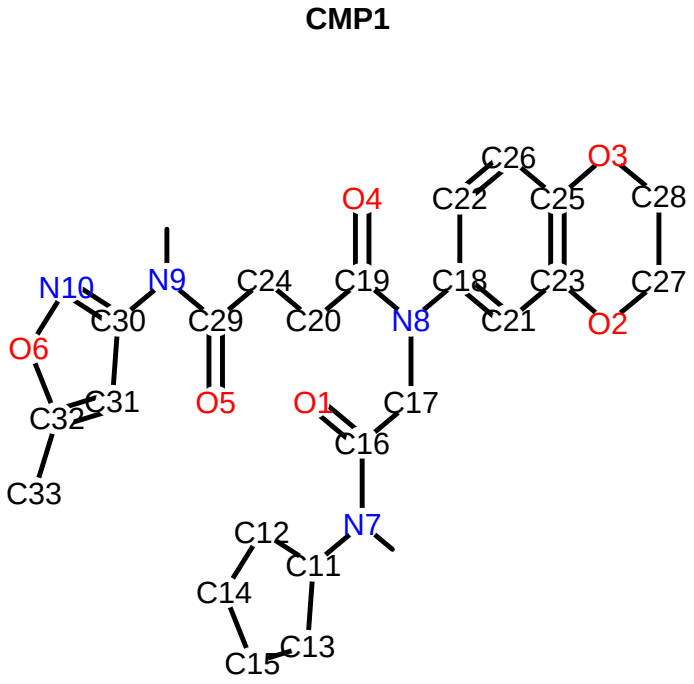
The following example returns all possible rings in a *list*. The argument `upper` allows to specify an upper length limit for rings. Choosing smaller length limits will reduce the search space resulting in shortened compute times. Note: each ring is represented by a character vector of atom symbols that are numbered by their position in the atom block of the corresponding *SDF/SDFset* object.

```
> rings(sdfset[1], upper=Inf, type="all", arom=FALSE, inner=FALSE)
```

```
$ring1
[1] "N_10" "O_6"  "C_32" "C_31" "C_30"

$ring2
[1] "C_12" "C_14" "C_15" "C_13" "C_11"

$ring3
[1] "C_23" "O_2"  "C_27" "C_28" "O_3"  "C_25"

$ring4
[1] "C_23" "C_21" "C_18" "C_22" "C_26" "C_25"

$ring5
 [1] "O_3"  "C_28" "C_27" "O_2"  "C_23" "C_21" "C_18" "C_22" "C_26" "C_25"
```

For visual inspection, the corresponding compound structure can be plotted with the same atom numbers as the rings:

```
> plot(sdfset[1], print=FALSE, atomnum=TRUE, no_print_atoms="H")
```



Aromaticity information of the rings can be returned in a logical vector by setting `arom=TRUE`:

```
> rings(sdfset[1], upper=Inf, type="all", arom=TRUE, inner=FALSE)
```

```
$RINGS
$RINGS$ring1
[1] "N_10" "O_6"  "C_32" "C_31" "C_30"

$RINGS$ring2
[1] "C_12" "C_14" "C_15" "C_13" "C_11"

$RINGS$ring3
[1] "C_23" "O_2"  "C_27" "C_28" "O_3"  "C_25"

$RINGS$ring4
[1] "C_23" "C_21" "C_18" "C_22" "C_26" "C_25"

$RINGS$ring5
 [1] "O_3"  "C_28" "C_27" "O_2"  "C_23" "C_21" "C_18" "C_22" "C_26" "C_25"
```

```
$AROMATIC
ring1 ring2 ring3 ring4 ring5
 TRUE FALSE FALSE  TRUE FALSE
```

Return rings with no more than 6 atoms that are also aromatic:

```
> rings(sdfset[1], upper=6, type="arom", arom=TRUE, inner=FALSE)

$AROMATIC_RINGS
$AROMATIC_RINGS$ring1
[1] "N_10" "O_6"  "C_32" "C_31" "C_30"

$AROMATIC_RINGS$ring4
[1] "C_23" "C_21" "C_18" "C_22" "C_26" "C_25"
```

Count shortest possible rings and their aromaticity assignments by setting `type=count` and `inner=TRUE`. The inner (smallest possible) rings are identified by first computing all possible rings and then selecting only the inner rings. For more details, consult the help documentation with `?rings`.

```
> rings(sdfset[1:4], upper=Inf, type="count", arom=TRUE, inner=TRUE)

     RINGS AROMATIC
CMP1     4        2
CMP2     3        3
CMP3     4        2
CMP4     3        3
```

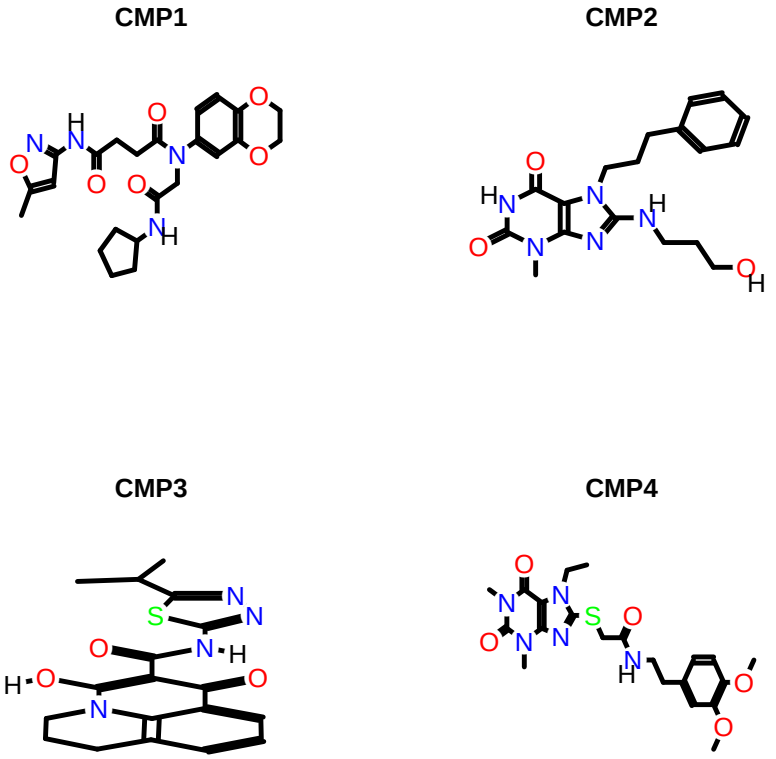
11 Rendering Chemical Structure Images

11.1 R Graphics Device

A new plotting function for compound structures has been added to the package recently. This function uses the native R graphics device for generating compound depictions. At this point this function is still in an experimental developmental stage but should become stable soon.

Plot compound Structures with R's graphics device:

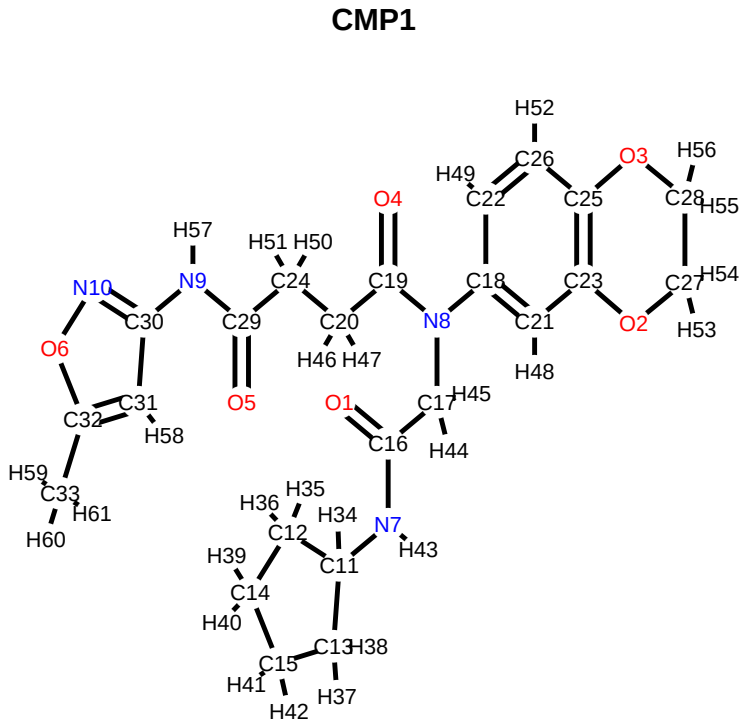
```
> data(sdfsample); sdfset <- sdfsample
> plot(sdfset[1:4], print=FALSE) # 'print=TRUE' returns SDF summaries
```



Customized plots:

```
> plot(sdfset[1:4], griddim=c(2,2), print_cid=letters[1:4], print=FALSE,
+      noHbonds=FALSE)
```

```
> plot(sdfset["CMP1"], atomnum = TRUE,  noHbonds=F , no_print_atoms = "",
+      atomcex=0.8, sub=paste("MW:", MW(sdfsample["CMP1"])), print=FALSE)
```



MW: 456.49162

In the above plot, the atom block position numbers in the SDF are printed next to the atom symbols (`atomnum = TRUE`). For more details, consult help documentation with `?plotStruc` or `?plot`.

11.2 Online with ChemMine Tools

Alternatively, one can visualize compound structures with a standard web browser using the online ChemMine Tools service. The service allows to display other information next to the structures using the extra argument of the `sdf.visualize` function. The following examples demonstrate, how one can plot and annotate structures by passing on extra data as vector of character strings, matrices or lists.

Plot structures using web service ChemMine Tools:

```
> sdf.visualize(sdfset[1:4])
```

Add extra annotation as *vector*:

```
> sdf.visualize(sdfset[1:4], extra=month.name[1:4])
```

Add extra annotation as *matrix*:

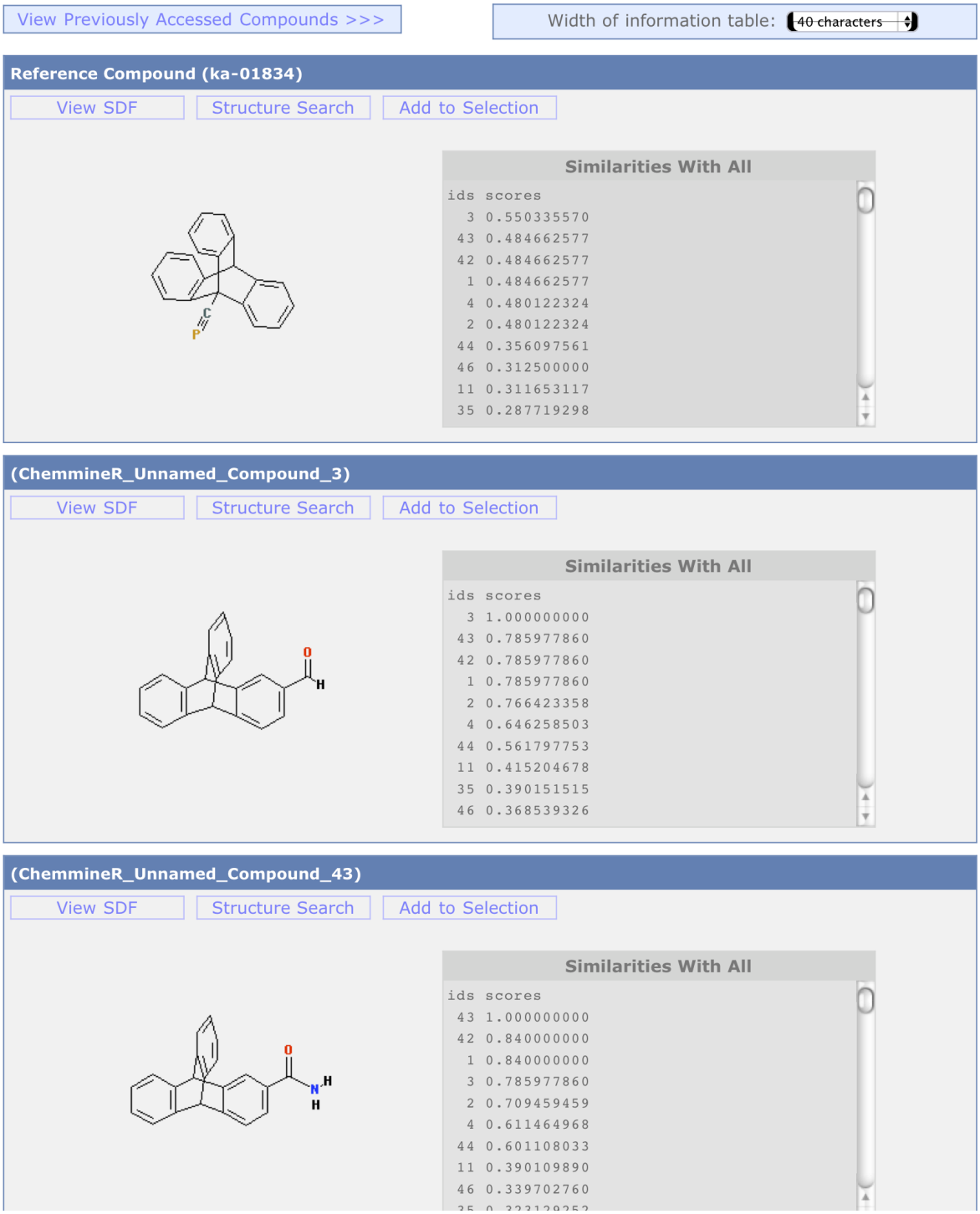


Figure 3: Visualization webpage created by calling `sdf.visualize`.

```
> extra <- apply(propma[1:4,], 1, function(x)
+          data.frame(Property=colnames(propma), Value=x))
> sdf.visualize(sdfset[1:4], extra=extra)
```

Add extra annotation as *list*:

```
> sdf.visualize(sdfset[1:4], extra=bondblock(sdfset[1:4]))
```

12 Similarity Comparisons and Searching

12.1 AP/APset Classes for Storing Atom Pair Descriptors

The function `sdf2ap` computes atom pair descriptors for one or many compounds (Carhart et al., 1985; Chen and Reynolds, 2002). It returns a searchable atom pair database stored in a container of class *APset*, which can be used for structural similarity searching and clustering. As similarity measure, the Tanimoto coefficient or related coefficients can be used. An *APset* object consists of one or many *AP* entries each storing the atom pairs of a single compound. Note: the deprecated `cmp.parse` function is still available which also generates atom pair descriptor databases, but directly from an SD file. Since the latter function is less flexible it may be discontinued in the future.

Generate atom pair descriptor database for searching:

```
> ap <- sdf2ap(sdfset[[1]]) # For single compound
> ap

An instance of "AP"
<<atom pairs>>
53688190976 53688190977 53688190978 53688190979 53688190980 ... length: 528

> apset <- sdf2ap(sdfset) # For many compounds.

> view(apset[1:4])

$`650001`
An instance of "AP"
<<atom pairs>>
53688190976 53688190977 53688190978 53688190979 53688190980 ... length: 528

$`650002`
An instance of "AP"
<<atom pairs>>
53688190976 53688190977 53688190978 53688190979 53689239552 ... length: 325

$`650003`
An instance of "AP"
<<atom pairs>>
52615496704 53688190976 53688190977 53689239552 53697627136 ... length: 325
```

```
$`650004`
An instance of "AP"
<<atom pairs>>
52617593856 52618642432 52619691008 52619691009 52628079616 ... length: 496
```

Return main components of APset objects:

```
> cid(apset[1:4]) # Compound IDs
> ap(apset[1:4]) # Atom pair descriptors
> db.explain(apset[1]) # Return atom pairs in human readable format
```

Coerce APset to other objects:

```
> apset2descdb(apset) # Returns old list-style AP database
> tmp <- as(apset, "list") # Returns list
> as(tmp, "APset") # Converts list back to APset
```

12.2 Large SDF and Atom Pair Databases

When working with large data sets it is often desirable to save the *SDFset* and *APset* containers as binary R objects to files for later use. This way they can be loaded very quickly into a new R session without recreating them every time from scratch.

Save and load of *SDFset* and *APset* containers:

```
> save(sdfset, file = "sdfset.rda", compress = TRUE)
> load("sdfset.rda")
> save(apset, file = "apset.rda", compress = TRUE)
> load("apset.rda")
```

12.3 Pairwise Compound Comparisons with Atom Pairs

The `cmp.similarity` function computes the atom pair similarity between two compounds using the Tanimoto coefficient as similarity measure. The coefficient is defined as $c/(a+b+c)$, which is the proportion of the atom pairs shared among two compounds divided by their union. The variable c is the number of atom pairs common in both compounds, while a and b are the numbers of their unique atom pairs.

```
> cmp.similarity(apset[1], apset[2])

[1] 0.2637037

> cmp.similarity(apset[1], apset[1])

[1] 1
```

12.4 Pairwise Compound Comparisons with PubChem Fingerprints

The `fpSim` function computes the Tanimoto coefficients for pairwise comparisons of binary fingerprints. For this data type, c is the number of "on-bits" common in both compounds, and a and b are the numbers of their unique "on-bits". Currently, the PubChem fingerprints need to be provided (here PubChem's SD files) and cannot be computed from scratch in *ChemmineR*. The PubChem fingerprint specifications can be loaded with `data(pubchemFPencoding)`.

Convert base 64 encoded PubChem fingerprints to character vector or binary matrix:

```
> cid(sdfset) <- sdfid(sdfset)
> fpset <- fp2bit(x=sdfset, type=1)
> fpset <- fp2bit(x=sdfset, type=2)
```

Pairwise compound structure comparisons:

```
> fpSim(x=fpset[1,], y=fpset[2,])
```

```
[1] 0.5344828
```

12.5 Similarity Searching with Atom Pairs

The `cmp.search` function searches an atom pair database for compounds that are similar to a query compound. The following example returns a data frame where the rows are sorted by the Tanimoto similarity score (best to worst). The first column contains the indices of the matching compounds in the database. The argument cutoff can be a similarity cutoff, meaning only compounds with a similarity value larger than this cutoff will be returned; or it can be an integer value restricting how many compounds will be returned. When supplying a cutoff of 0, the function will return the similarity values for every compound in the database.

```
> cmp.search(apset, apset["650065"], type=3, cutoff = 0.3, quiet=TRUE)
```

```
  index    cid    scores
1    61 650066 1.0000000
2    60 650065 1.0000000
3    67 650072 0.3389831
4    11 650011 0.3190608
5    15 650015 0.3184524
6    86 650092 0.3154270
7    64 650069 0.3010279
```

Alternatively, the function can return the matches in form of an index or a named vector if the `type` argument is set to `1` or `2`, respectively.

```
> cmp.search(apset, apset["650065"], type=1, cutoff = 0.3, quiet=TRUE)
```

```
[1] 61 60 67 11 15 86 64
```

```
> cmp.search(apset, apset["650065"], type=2, cutoff = 0.3, quiet=TRUE)
```

```
   650066    650065    650072    650011    650015    650092    650069
1.0000000 1.0000000 0.3389831 0.3190608 0.3184524 0.3154270 0.3010279
```

12.6 Similarity Searching with PubChem Fingerprints

Similarly, the `fpSim` function provides search functionality for PubChem fingerprints:

```
> fpSim(x=fpset["650065",], y=fpset)[1:6] # x is query and y is fingerprint database
```

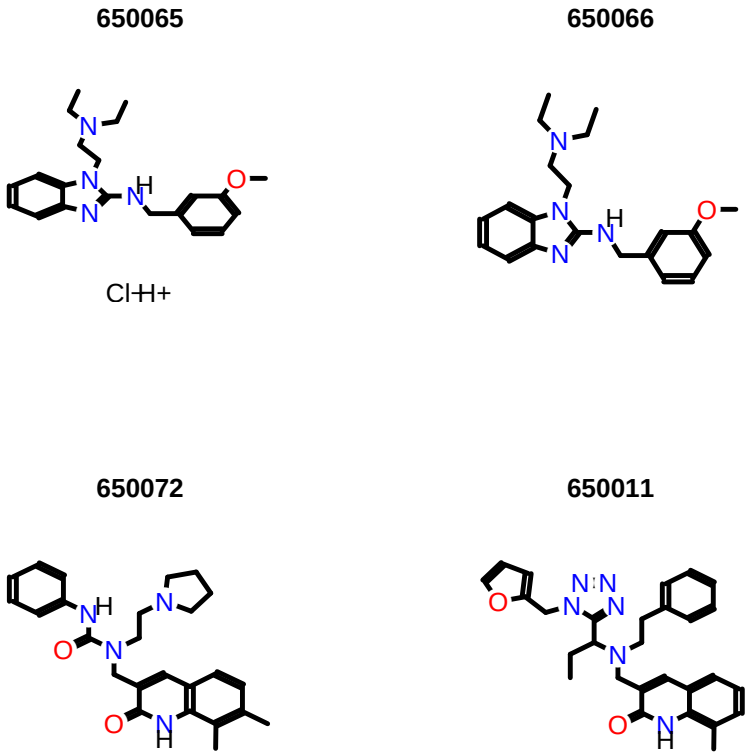
```
   650065    650066    650035    650019    650012    650046
1.0000000 0.9944444 0.7422680 0.7420814 0.7216981 0.7129187
```

12.7 Visualize Similarity Search Results

The `cmp.search` function allows to visualize the chemical structures for the search results. Similar but more flexible chemical structure rendering functions are `plot` and `sdf.visualize` described above. By setting the visualize argument in `cmp.search` to `TRUE`, the matching compounds and their scores can be visualized with a standard web browser. Depending on the `visualize.browse` argument, an URL will be printed or a webpage will be opened showing the structures of the matching compounds along with their scores.

View similarity search results in R's graphics device:

```
> cid(sdfset) <- cid(apset) # Assure compound name consistency among objects.
> plot(sdfset[names(cmp.search(apset, apset["650065"], type=2, cutoff=4,
+      quiet=TRUE))], print=FALSE)
```



View results online with Chemmine Tools:

```
> similarities <- cmp.search(apset, apset[1], type=3, cutoff = 10)
> sdf.visualize(sdfset[similarities[,1]], extra=similarities[,3])
```

13 Clustering

13.1 Clustering Identical or Very Similar Compounds

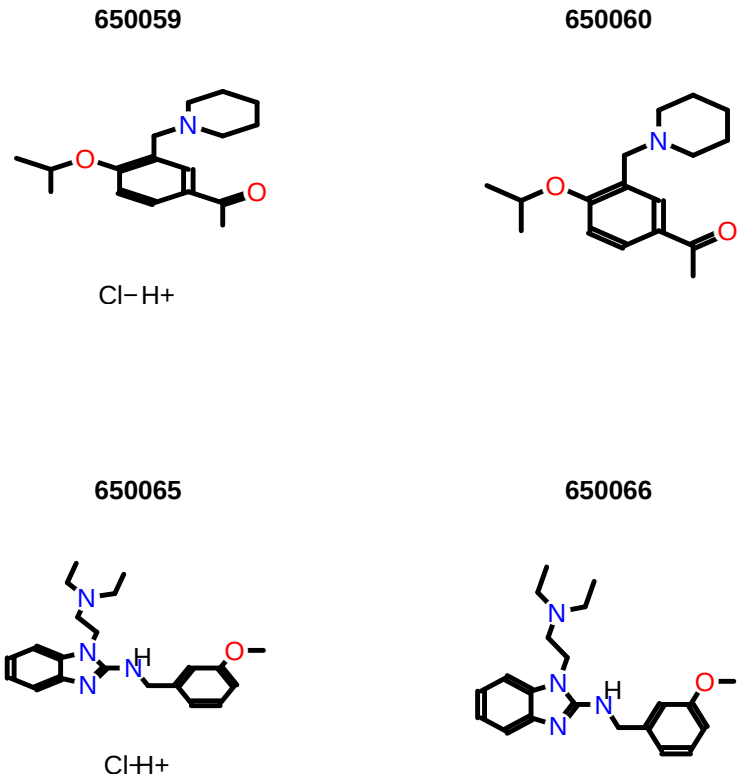
Often it is of interest to identify very similar or identical compounds in a compound set. The `cmp.duplicated` function can be used to quickly identify very similar compounds in atom pair sets, which will be frequently, but not necessarily, identical compounds.

Identify compounds with identical AP sets:

```
> cmp.duplicated(apset, type=1)[1:4] # Returns AP duplicates as logical vector

[1] FALSE FALSE FALSE FALSE

> cmp.duplicated(apset, type=2)[1:4,] # Returns AP duplicates as data frame

     ids CLSZ_100 CLID_100
1 650082        1        1
2 650059        2        2
3 650060        2        2
4 650010        1        3
```

Plot the structure of two pairs of duplicates:

```
> plot(sdfset[c("650059","650060", "650065", "650066")], print=FALSE)
```



Remove AP duplicates from SDFset and APset objects:

```
> apdups <- cmp.duplicated(apset, type=1)
> sdfset[which(!apdups)]; apset[which(!apdups)]

An instance of "SDFset" with 96 molecules

An instance of "APset" with 96 molecules
```

Alternatively, one can identify duplicates via other descriptor types if they are provided in the data block of an imported SD file. For instance, one can use here fingerprints, InChI, SMILES or other molecular representations. The following examples show how to enumerate by identical InChI strings, SMILES strings and molecular formula, respectively.

```
> count <- table(datablocktag(sdfset, tag="PUBCHEM_NIST_INCHI"))
> count <- table(datablocktag(sdfset, tag="PUBCHEM_OPENEYE_CAN_SMILES"))
> count <- table(datablocktag(sdfset, tag="PUBCHEM_MOLECULAR_FORMULA"))
> count[1:4]

 C10H9FN2O2S   C11H12N4OS    C11H13NO4 C12H11ClN2OS
           1            1            1            1
```

13.2 Binning Clustering

Compound libraries can be clustered into discrete similarity groups with the binning clustering function `cmp.cluster`. The function requires as input an atom pair descriptor database as well as a similarity threshold. The binning clustering result is returned in form of a data frame. Single linkage is used for cluster joining. The function calculates the required compound-to-compound distance information on the fly, while a memory-intensive distance matrix is only created upon user request via the `save.distances` argument (see below).

Because an optimum similarity threshold is often not known, the `cmp.cluster` function can calculate cluster results for multiple cutoffs in one step with almost the same speed as for a single cutoff. This can be achieved by providing several cutoffs under the cutoff argument. The clustering results for the different cutoffs will be stored in one data frame.

One may force the `cmp.cluster` function to calculate and store the distance matrix by supplying a file name to the `save.distances` argument. The generated distance matrix can be loaded and passed on to many other clustering methods available in R, such as the hierarchical clustering function `hclust` (see below).

If a distance matrix is available, it may also be supplied to `cmp.cluster` via the `use.distances` argument. This is useful when one has a pre-computed distance matrix either from a previous call to `cmp.cluster` or from other distance calculation subroutines.

Single-linkage binning clustering with one or multiple cutoffs:

```
> clusters <- cmp.cluster(db=apset, cutoff = c(0.65, 0.5, 0.4), quiet = TRUE)

sorting result...

> clusters[1:4,]

      ids CLSZ_0.65 CLID_0.65 CLSZ_0.5 CLID_0.5 CLSZ_0.4 CLID_0.4
48 650049         2        48        2       48        2       48
49 650050         2        48        2       48        2       48
54 650059         2        54        2       54        2       54
55 650060         2        54        2       54        2       54
```

Return cluster size distributions for each cutoff:

```
> cluster.sizestat(clusters, cluster.result=1)

  cluster size count
1            1    90
2            2     5

> cluster.sizestat(clusters, cluster.result=2)

  cluster size count
1            1    80
2            2    10

> cluster.sizestat(clusters, cluster.result=3)
```

```
  cluster size count
1            1    68
2            2     8
3            4     1
4           12     1
```

Enforce calculation of distance matrix:

```
> clusters <- cmp.cluster(db=apset, cutoff = c(0.65, 0.5, 0.3),
+                         save.distances="distmat.rda")
>    # Saves distance matrix to file "distmat.rda" in current working directory.
> load("distmat.rda") # Loads distance matrix.
```

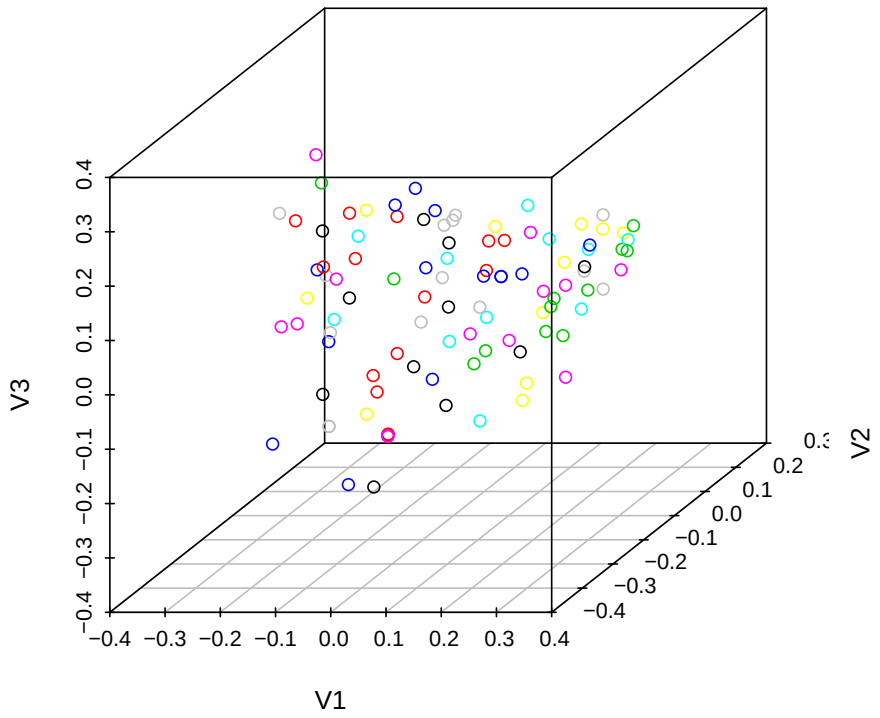
13.3 Multi-Dimensional Scaling (MDS)

To visualize and compare clustering results, the `cluster.visualize` function can be used. The function performs Multi-Dimensional Scaling (MDS) and visualizes the results in form of a scatter plot. It requires as input an *APset*, a clustering result from `cmp.cluster`, and a cutoff for the minimum cluster size to consider in the plot. To help determining a proper cutoff size, the `cluster.sizestat` function is provided to generate cluster size statistics.
MDS clustering and scatter plot:

```
> cluster.visualize(apset, clusters, size.cutoff=2, quiet = TRUE)
>    # Color codes clusters with at least two members.
> cluster.visualize(apset, clusters, quiet = TRUE) # Plots all items.
```

Create a 3D scatter plot of MDS result:

```
> library(scatterplot3d)
> coord <- cluster.visualize(apset, clusters, size.cutoff=1, dimensions=3, quiet=TRUE)
> scatterplot3d(coord)
```



Interactive 3D scatter plot with Open GL (graphics not evaluated here):

```
> library(rgl)
> rgl.open(); offset <- 50; par3d(windowRect=c(offset, offset, 640+offset, 640+offset))
> rm(offset); rgl.clear(); rgl.viewpoint(theta=45, phi=30, fov=60, zoom=1)
> spheres3d(coord[,1], coord[,2], coord[,3], radius=0.03, color=coord[,4], alpha=1,
+ shininess=20); aspect3d(1, 1, 1)
> axes3d(col='black'); title3d("", "", "", "", "", col='black'); bg3d("white")
>    # To save a snapshot of the graph, one can use the command rgl.snapshot("test.png").
```

13.4 Clustering with Other Algorithms

ChemmineR allows the user to take advantage of the wide spectrum of clustering utilities available in R. An example on how to perform hierarchical clustering with the hclust function is given below.

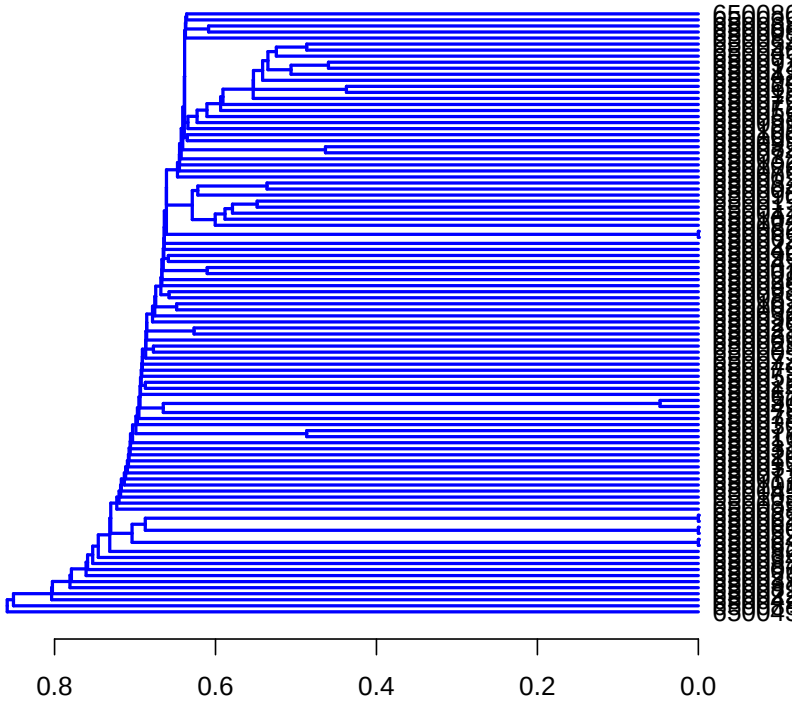
Create atom pair distance matrix:

```
> dummy <- cmp.cluster(db=apset, cutoff=0, save.distances="distmat.rda", quiet=TRUE)

sorting result...
```

```
> load("distmat.rda")
```

Hierarchical clustering with `hclust`:

```
> hc <- hclust(as.dist(distmat), method="single")
> hc[["labels"]] <- cid(apset) # Assign correct item labels
> plot(as.dendrogram(hc), edgePar=list(col=4, lwd=2), horiz=T)
```

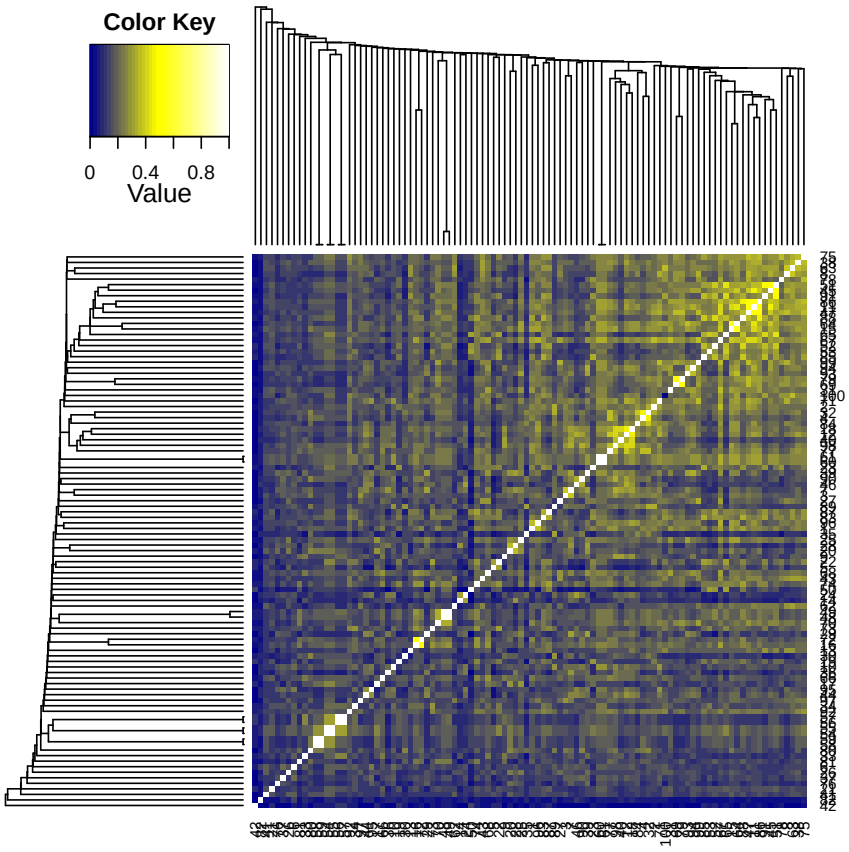


Instead of atom pairs one can use PubChem's fingerprints for clustering:

```
> simMA <- sapply(rownames(fpset), function(x) fpSim(x=fpset[x,], fpset))
> hc <- hclust(as.dist(simMA), method="single")
> plot(as.dendrogram(hc), edgePar=list(col=4, lwd=2), horiz=TRUE)
```

Plot dendrogram with heatmap (here similarity matrix):

```
> library(gplots)
> heatmap.2(1-distmat, Rowv=as.dendrogram(hc), Colv=as.dendrogram(hc),
+           col=colorpanel(40, "darkblue", "yellow", "white"),
+           density.info="none", trace="none")
```



14 Searching PubChem

14.1 Get Compounds from PubChem by Id

The function `getIds` accepts one or more numeric PubChem compound ids and downloads the corresponding compounds from PubChem Power User Gateway (PUG) returning results in an *SDFset* container. The ChemMine Tools web service is used as an intermediate, to translate queries from plain HTTP POST to a PUG SOAP query.

Fetch 2 compounds from PubChem:

```
> compounds <- getIds(c(111,123))
> compounds
```

14.2 Search a SMILES Query in PubChem

The function `searchString` accepts one SMILES string (Simplified Molecular Input Line Entry Specification) and performs a >0.95 similarity PubChem fingerprint search, returning the hits in an *SDFset* container. The ChemMine Tools web service is used as an intermediate, to

translate queries from plain HTTP POST to a PubChem Power User Gateway (PUG) query.

Search a SMILES string on PubChem:

```
> compounds <- searchString("CC(=O)OC1=CC=CC=C1C(=O)O")
> compounds
```

14.3 Search an SDF Query in PubChem

The function `searchSim` performs a PubChem similarity search just like `searchString`, but accepts a query in an *SDFset* container. If the query contains more than one compound, only the first is searched.

Search an *SDFset* container on PubChem:

```
> data(sdfsample); sdfset <- sdfsample[1]
> compounds <- searchSim(sdfset)
> compounds
```

15 Format Interconversions

The `sdf2smiles` and `smiles2sdf` functions provide format interconversion between SMILES strings (Simplified Molecular Input Line Entry Specification) and *SDFset* containers. Currently these functions are limited to a single compound at a time.

Convert an *SDFset* container to a SMILES *character* string:

```
> data(sdfsample); sdfset <- sdfsample[1]
> smiles <- sdf2smiles(sdfset)
> smiles
```

Convert a SMILES *character* string to an *SDFset* container:

```
> sdf <- smiles2sdf("CC(=O)OC1=CC=CC=C1C(=O)O\tname")
> view(sdf)
```

These functions require internet connectivity, as they rely on the ChemMine Tools web service for conversion with the Open Babel Open Source Chemistry Toolbox.

16 Version Information

```
> sessionInfo()
```

```
R version 2.15.1 (2012-06-22)
Platform: i386-apple-darwin9.8.0/i386 (32-bit)

locale:
[1] C
```

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

other attached packages:
[1] gplots_2.11.0        MASS_7.3-20          KernSmooth_2.23-8
[4] caTools_1.13         bitops_1.0-4.1       gdata_2.11.0
[7] gtools_2.7.0         scatterplot3d_0.3-33 ChemmineR_2.8.1

loaded via a namespace (and not attached):
[1] RCurl_1.91-1 tools_2.15.1
```

References

T W Backman, Y Cao, and T Girke. ChemMine tools: an online service for analyzing and clustering small molecules. *Nucleic Acids Res*, 39(Web Server issue):486–491, Jul 2011. doi: 10.1093/nar/gkr320. URL `http://www.hubmed.org/display.cgi?uids=21576229`.

Y Cao, A Charisi, L C Cheng, T Jiang, and T Girke. ChemmineR: a compound mining framework for R. *Bioinformatics*, 24(15):1733–1734, Aug 2008. doi: 10.1093/bioinformatics/btn307. URL `http://www.hubmed.org/display.cgi?uids=18596077`.

R.E. Carhart, D.H. Smith, and R. Venkataraghavan. Atom pairs as molecular features in structure-activity studies: definition and applications. *Journal of Chemical Information and Computer Sciences*, 25(2):64–73, 1985.

X. Chen and C.H. Reynolds. Performance of Similarity Measures in 2D Fragment-Based Similarity Searching: Comparison of Structural Descriptors and Similarity Coefficients. *Journal of Chemical Information and Computer Sciences*, 42(6):1407–1414, 2002.

Th. Hanser, Ph. Jauffret, and G. Kaufmann. A new algorithm for exhaustive ring perception in a molecular graph. *Journal of Chemical Information and Computer Sciences*, 36(6): 1146–1152, 1996. doi: 10.1021/ci960322f. URL `http://pubs.acs.org/doi/abs/10.1021/ci960322f`.