ChemmineR: A Compound Mining Toolkit for Chemical Genomics in R

Yiqun Cao
Anna Charisi
Thomas Girke

April 10, 2009

Introduction

The *ChemmineR* package includes functions for calculating atom pair descriptors of chemical compounds (Carhart et al., 1985; Chen and Reynolds, 2002; Guha, 2007), structural similarity searching, clustering of compound libraries, and visualization of cluster results and chemical structures. These utilities are often important for the assembly of compound libraries and typical analysis routines of chemical genomics screening projects. An overview of all functions is illustrated in Fig. 1. The `cmp.parse` function accepts an SD file (SDF) of a whole library, parses it, and generates a descriptor database for all the compounds in the library. Similarly, the `cmp.parse1` function accepts an SDF for a single compound, parses it, and generates a descriptor vector for that compound. The `cmp.similarity` function computes atom pair-based similarities between two compounds using by default the Tanimoto coefficient as similarity measure. Searching for compounds in a library that are similar to a query structure can be accomplished with the `cmp.search` function. The `cmp.cluster` , `cluster.sizestat`, and `cluster.visualize` functions together allow binning clustering of compounds in a parsed library. The `sdf.subset` function provides utiliies for managing and subsetting libraries in SDF format, while the `sdf.visualize` function converts their compounds into images of chemical structures on HTML pages. *ChemmineR* integrates well with the online ChemMine (Girke et al., 2005) portal which provides access to extensive compound annotations and web-based chemoinformatic tools.

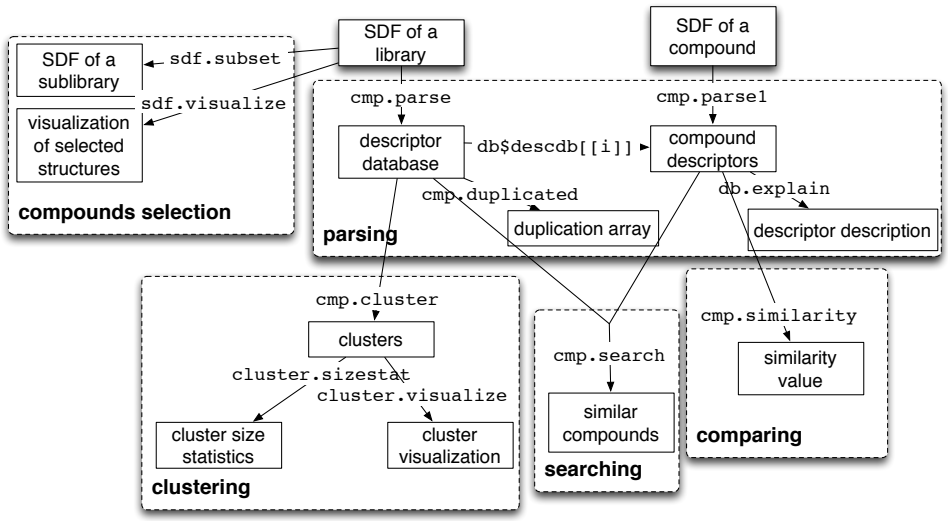


Figure 1: Overview of functions provided in the *ChemmineR* package.

Installing the Package

Users can download the package from the download page (`http://bioweb.ucr.edu/ChemMineV2/chemminer/download`) for the OS of their choice, and then use the `install.packages("...", repos=NULL)` command to install the package. More detailed instructions for Mac, Linux and Windows can be found in the online manual (`http://bioweb.ucr.edu/ChemMineV2/chemminer/tutorial`).

Loading the Package and Its Documentation

The package can be loaded with the standard `library` function:

```
> library(ChemmineR)
```

The following command lists all functions available in the package:

```
> library(help = ChemmineR)
> options(chemminer.max.upload = 100)
```

The package contains a PDF manual, which can be loaded in the standard way:

```
> vignette("ChemmineR")
```

Compound Database Import from SDF

The `cmp.parse` function imports SDFs containing the data for one or many compounds. It returns a searchable atom pair database, which can be used for structural similarity searching, clustering, and SDF manipulation. The only argument of this function is the path (or URL) to the file containing the SDF information.

```
> library(ChemmineR)
> db <- cmp.parse("http://bioweb.ucr.edu/ChemMineV2/static/example_db.sdf",
+     quiet = TRUE)

counting number of compounds in sdf...
129  compounds found

you can use save(..., file='...', compress=TRUE) to save the database
```

The database can be saved as an R-specific binary file for faster loading in the future.

```
> save(db, file = "compounddb.rda", compress = TRUE)
```

The `load` function loads the database back into R without repeating the time-consuming descriptor generation step.

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

Single Compound Import from SDF

The `cmp.parse1` function will parse an SDF for a single compound. Similarly as before, the only argument required is the path (or URL) to the SDF.

```
> query.url <- "http://bioweb.ucr.edu/ChemMineV2/compound/Aurora/b32:NNQS2MBRHAZTI===
> query <- cmp.parse1(query.url)
```

Descriptor Database Content

The descriptors of compounds are stored as numeric vectors in a list object along with available annotation information about the database. You may skip this section if you are not interested in internals of descriptor database.

The `cmp.parse1` function parses the SDF of a single compound, generates the descriptors and stores them in a numeric vector. Each entry of the vector is a descriptor for this compound.

```
> class(query)

[1] "numeric"
```

In contrast to this, the `cmp.parse` function generates a list object with four components.

```
> names(db)

[1] "descdb"  "cids"    "sdfsegs" "source"
```

The `descdb` component is a list. Each entry of the list is a vector of descriptors of one compound.

```
> class(db$descdb)

[1] "list"

> class(db$descdb[[1]])

[1] "numeric"
```

The `db.explain` function returns the descriptors in a human readable format. A single descriptor can be returned like this:

```
> db.explain(query[1])

[1] "C [2 neighbor(s),1 pi electrons]<---1--->C [2 neighbor(s),1 pi electrons]"

> db.explain(db$descdb[[1]][1])

[1] "C [2 neighbor(s),1 pi electrons]<---1--->C [2 neighbor(s),1 pi electrons]"
```

The same is possible for multiple descriptors at once.

```
> db.explain(db$descdb[[1]][1:5])

[1] "C [2 neighbor(s),1 pi electrons]<---1--->C [2 neighbor(s),1 pi electrons]"
[2] "C [2 neighbor(s),1 pi electrons]<---1--->C [2 neighbor(s),1 pi electrons]"
[3] "C [2 neighbor(s),1 pi electrons]<---1--->C [2 neighbor(s),1 pi electrons]"
[4] "C [2 neighbor(s),1 pi electrons]<---1--->C [2 neighbor(s),1 pi electrons]"
[5] "C [2 neighbor(s),1 pi electrons]<---1--->C [2 neighbor(s),1 pi electrons]"
```

Removing Duplicated Compounds

The `cmp.duplicated` function can be used to quickly identify and remove duplicated compounds in imported compound databases. In most cases the method will identify the duplicates correctly. However, users have to be aware that the atom pair algorithm will treat isomers, conformers and other smaller structural variants as identical compounds. If it is important to retain those variants in the data set then the function `cmp.duplicated` should not be used. The support of InChI stings will overcome this limitation in the future.

The function takes a descriptor database as the only required argument and returns the duplication information as a logical vector.

To demo this feature on the imported sample data set, one can create a duplication with the following command.

```
> db$descdb <- c(db$descdb, list(db$descdb[[1]]))
```

In the next step the duplication is identified with the `cmp.duplicated` function.

```
> dup <- cmp.duplicated(db)
> dup
```

```
  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [10] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [19] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [28] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [46] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [55] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [64] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [73] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [82] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [91] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[100] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[109] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[118] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[127] FALSE FALSE FALSE  TRUE
```

The `TRUE` entry in the returned logical vector indicates the duplication. It can be easily removed with the standard R subsetting syntax.

```
> db$descdb <- db$descdb[!dup]
```

In a real example one also needs to remove the duplications from the other database components.

```
> db$cids <- db$cids[!dup]
> db$sdfsegs <- db$sdfsegs[!dup]
```

Pairwise Compound Comparisons

The `cmp.similarity` function computes the atom pair similarity between two compounds using the Tanimoto coefficient as similarity measure. The Tanimoto coefficient between the atom pair descriptors of two compounds (CMP A and CMP B) is calculated here according to the following formula:

$$\text{Tanimoto coefficient} = c/(a+b+c)$$

a = count of atom pair descriptors in CMP A but not in CMP B
b = count of atom pair descriptors in CMP B but not in CMP A
c = count of atom pair descriptors shared by CMP A and CMP B

```
> similarity <- cmp.similarity(db$descdb[[1]], db$descdb[[2]])
> cat(paste("The similarity between compound 1 and 2 is ",
+     similarity, "\n", sep = ""))

The similarity between compound 1 and 2 is 0.70945945945946

> similarity <- cmp.similarity(db$descdb[[1]], query)
> cat(paste("The similarity between compound 1 and the query is ",
+     similarity, "\n", sep = ""))

The similarity between compound 1 and the query is 0.484662576687117
```

With the `cmp.similarity` function one can easily design custom search subroutines similar to the one introduced in the next section.

Similarity Searching

The `cmp.search` function searches an atom pair database for compounds that are similar to a query compound.

```
> cmp.search(db, query, cutoff = 0.4, return.score = TRUE,
+     quiet = TRUE)
```

```
  ids    scores
1   3 0.5503356
2  43 0.4846626
3  42 0.4846626
4   1 0.4846626
5   4 0.4801223
6   2 0.4801223
```

The function returns a data frame where the rows are sorted by 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. If the argument `return.score` is set to `FALSE`, then the function will return a vector of indices rather than a data frame. When supplying a cutoff of 0, the function will return the similarity values for every compound in the database.

```
> similarities <- cmp.search(db, db[[1]][[1]], cutoff = 0,
+     return.score = TRUE, quiet = TRUE)
```

The `cmp.search` function allows to visualize the chemical structure images for the search results. A similar but more flexible chemical structure rendering function (`sdf.visualize`) is described later in this manual. By setting the `visualize` argument in `cmp.search` to `TRUE`, the matching compounds and their scores can be visualized with a standard web browser on the online ChemMine interface. 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.

```
> similarities <- cmp.search(db, query, cutoff = 10,
+     quiet = TRUE, visualize = TRUE, visualize.browse = FALSE)
```

Setting the `visualize.browse` argument to `TRUE` will automatically open the webpage in the default browser.

The query structure can also be displayed on the visualization webpage by supplying the SDF of the query in a character string or providing its file name or URL. For example,

```
> similarities <- cmp.search(db, query, cutoff = 10,
+     quiet = TRUE, visualize = TRUE, visualize.browse = TRUE,
+     visualize.query = query.url)
```

This will read the SDF provided by `query.url`, and display it as a "reference compound" at the top of the page. Part of the screenshot of the resulting output is shown in Fig. 2. A live demo is also available and linked from the online version of this manual (`http://bioweb.ucr.edu/ChemMineV2/chemminer/tutorial`).

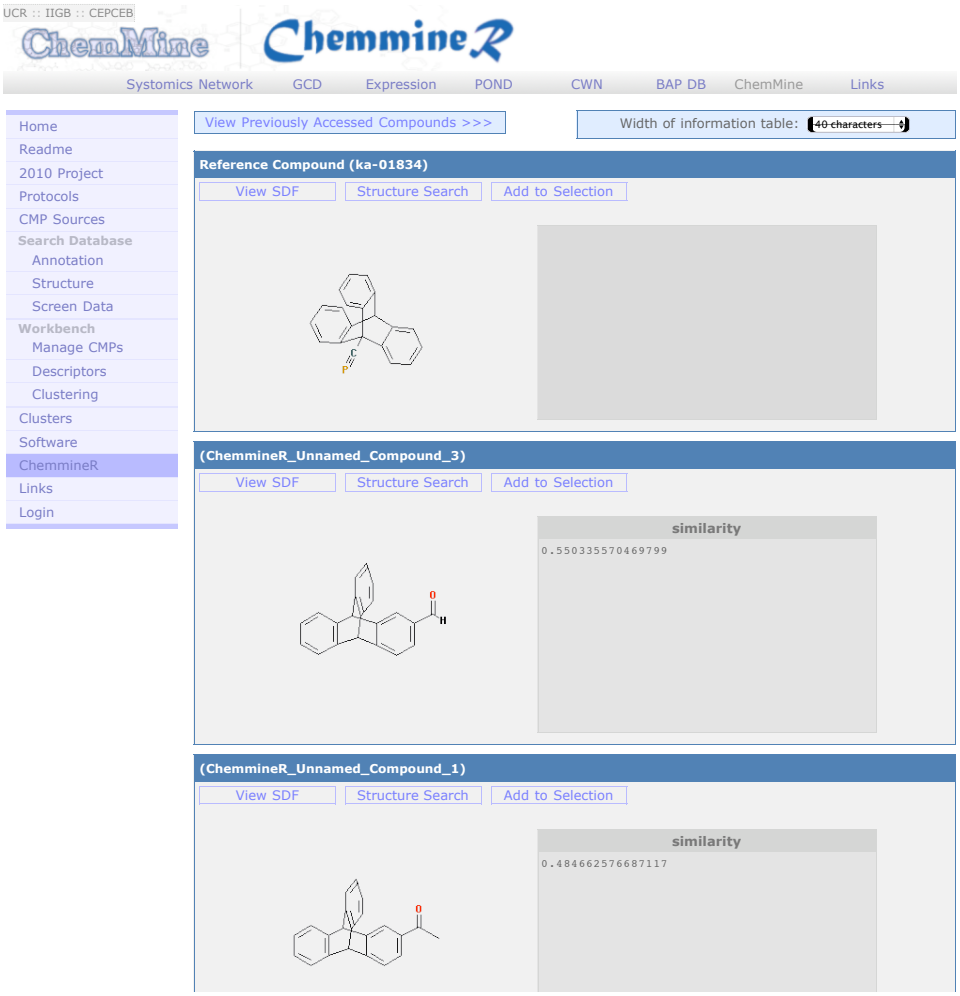


Figure 2: `cmp.search` can automatically upload the structures and scores of matching compounds to ChemMine for visualization.

Any information uploaded to *ChemMine* by *ChemmineR* is kept private and secure using a highly randomized URL. The visualization pages can be shared with colleagues by providing the corresponding URLs.

Rendering Chemical Structure Images

Internally, the similarity search function uses the `sdf.visualize` function to send compounds to ChemMine for structure visualization. The same function can be used to send any custom combination of compounds for visualization on ChemMine along with complex annotation and activity information. The function accepts a database and a vector of compound indices. The following example performs first a similarity search to obtain a vector of indices.

```
> indices <- cmp.search(db, query, cutoff = 10,
+     quiet = TRUE)

> url <- sdf.visualize(db, indices, browse = TRUE,
+     quiet = TRUE)

> url

[1] "http://bioweb.ucr.edu/ChemMineV2/chemminer/viewsdfs?ref=86bb9a295656e1182101b7a7
```

The URL stored in the `url` object points to a webpage that shows the structures of the compounds. If the `browse` argument is set to `TRUE`, then the default browser will open automatically.

In addition, one can display other information next to the structures using the `extra` argument. In the following example, a vector of character strings is assigned to `extra`, and its entries are displayed next to corresponding chemical structures.

```
> extra.info <- paste("Matching compound #", 1:length(indices),
+     sep = "")
> names(extra.info) <- rep("Note", length(indices))

> url <- sdf.visualize(db, indices, browse = TRUE,
+     quiet = TRUE, extra = extra.info)

> url

[1] "http://bioweb.ucr.edu/ChemMineV2/chemminer/viewsdfs?ref=34c10198f44fb5373a51110c
```

The function also allows to list a reference compound at the top of the page. The user supplies the SDF of this reference compound in form of a character string or a file. Annotation information can also be displayed next to the reference structure.

```
> url <- sdf.visualize(db, indices, browse = TRUE,
+     quiet = TRUE, reference.sdf = query.url, reference.note = "Reference Compound f

> url

[1] "http://bioweb.ucr.edu/ChemMineV2/chemminer/viewsdfs?ref=a42ef52ed3374ac9ac5f888c
```

It is also possible to display more complex tabular data next to each compound by providing a list of data frames. To demonstrate this utility, the following example creates such a list of data frames via a similarity search. Each data frame is then displayed next to the corresponding compound. The screenshot of the resulting output is shown in Fig. 3.

To generate this output, first a similarity is performed using a cutoff of 0 to obtain the similarity values between the query compound and each of the compounds in the database.

```
> search.results <- cmp.search(db, query, cutoff = 0,
+     return.score = T, quiet = T)
```

The resulting data frame will be used as annotation table for the query compound. To provide a table name, one has to embed it into a list. If a table name is not required, then there is no need to generate the list object

```
> note.q <- list(search.results)
> names(note.q) <- "Similarities With All"
```

For each of the top 10 hits in the search result, we perform the same search to obtain the similarity values between the hit and all compounds in the database. This information will then be displayed next to the structures on the visualization page.

```
> indices <- search.results[1:10, 1]
> notes <- list()
> for (i in indices) {
+     search.results <- cmp.search(db, db$descdb[[i]],
+         0, return.score = T, quiet = T)
+     notes <- c(notes, list(search.results))
+ }
> names(notes) <- rep("Similarities With All", 10)
```

The following step displays the complex sample data set on ChemMine.

```
> url <- sdf.visualize(db, indices, extra = notes,
+     browse = T, reference.sdf = query.url, reference.note = note.q)
```

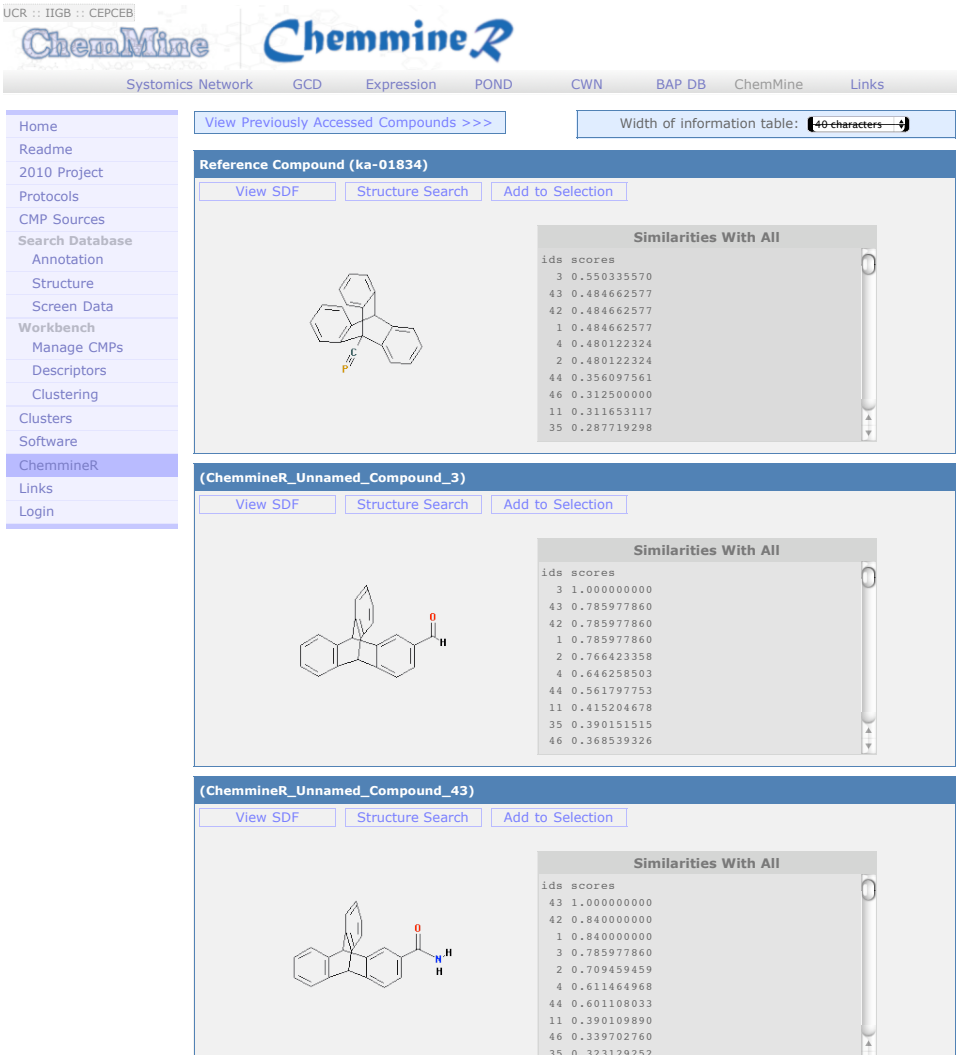


Figure 3: Visualization webpage created by calling `sdf.visualize`. This page shows the table information properly rendered and displayed next to the compound structures.

Note: the `sdf.visualize` function depends on the original SDF file from which the descriptor database has been generated. If the SDF file has been moved or altered then this step cannot be used.

Any information uploaded to *ChemMine* by *ChemmineR* is kept private and secure using a highly randomized URL. The visualization pages can be shared with colleagues by providing the corresponding URLs.

Subsetting SDF Batch Files

After identifying a subset of interesting compounds, one can generate an SDF for this subset of compounds using the `sdf.subset` function.

For example, one can perform a similarity search, and use the top 10 results for subsetting.

```
> indices <- cmp.search(db, query, cutoff = 10,
+     quiet = TRUE)
```

With the corresponding indices one can generate a custom SDF batch data set and store it in an external file.

```
> sdf <- sdf.subset(db, indices)
> cat(sdf, file = "matching.sdf")
```

One may also create a sub-database from a descriptor database using the related `db.subset` function.

```
> db_sub <- db.subset(db, indices)
```

Note: the `sdf.visualize` function depends on the original SDF file from which the descriptor database has been generated.

Binning Clustering

Compound libraries can be clustered into discrete similarity groups with the binning clustering function `cmp.cluster`. This binning clustering method is optimized for the typical compound library analysis routines for high-throughput screening projects. The algorithm uses single-linkage clustering to join compounds into similarity groups, where every member in a cluster shares with at least one other member a similarity value above a user-specified threshold. The algorithm is optimized for speed and memory efficiency by avoiding the calculation of an all-against-all distance matrix.

This is achieved by calculating on-the-fly only the distance values that are required in each clustering step. Because an optimum similarity threshold is often unknown, a series of binning clustering results can be calculated simultaneously for several user-specified thresholds. Cluster results for several thresholds can be calculated almost with the same speed as for a single threshold by issuing multiple clustering processes simultaneously, but calculating the required distances only once.

The function requires as input a 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).

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

sorting result...
```

The previous step clusters the compounds stored in `db` with a similarity cutoff of 0.65. In other words, if two compounds share a similarity of 0.65 or above, then they will be joined into the same cluster. The first 10 rows of the result data frame are shown here:

```
> clusters[1:10, ]

   ids CLSZ_0.65 CLID_0.65
1    1         7         1
2    2         7         1
3    3         7         1
4    4         7         1
42  42         7         1
43  43         7         1
44  44         7         1
14  14         3        14
17  17         3        14
18  18         3        14
```

The first column contains the compound IDs, the second the cluster size and third the cluster ID. The compound in cluster ID 1 can be returned with the following command:

```
> clusters[clusters[, 3] == 1, ]
```

```
   ids CLSZ_0.65 CLID_0.65
1    1         7         1
2    2         7         1
3    3         7         1
4    4         7         1
42  42         7         1
43  43         7         1
44  44         7         1
```

Similarly as above, one can visualize the chemical structures for a compound cluster of interest with the `sdf.visualize` function.

```
> ids <- clusters[clusters[, 3] == 23, 1]
> sdf.visualize(db, ids, browse = TRUE, quiet = TRUE)
```

Binning Clustering with Multiple Cutoffs

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. The clustering results for the different cutoffs will be stored in one data frame.

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

sorting result...
```

The first 10 rows of the generated cluster result data frame are:

```
> clusters[1:10, ]

   ids CLSZ_0.65 CLID_0.65 CLSZ_0.5 CLID_0.5
1    1         7         1        7        1
2    2         7         1        7        1
3    3         7         1        7        1
4    4         7         1        7        1
42  42         7         1        7        1
43  43         7         1        7        1
44  44         7         1        7        1
14  14         3        14        4       14
17  17         3        14        4       14
18  18         3        14        4       14
```

Cluster 14 obtained by the cutoff 0.65 contains the following compounds:

```
> clusters[clusters[, "CLID_0.65"] == 14, ]

   ids CLSZ_0.65 CLID_0.65 CLSZ_0.5 CLID_0.5
14  14         3        14        4       14
17  17         3        14        4       14
18  18         3        14        4       14
```

and cluster 14 from cutoff 0.5 contains:

```
> clusters[clusters[, "CLID_0.5"] == 14, ]

   ids CLSZ_0.65 CLID_0.65 CLSZ_0.5 CLID_0.5
14  14         3        14        4       14
17  17         3        14        4       14
18  18         3        14        4       14
16  16         1        16        4       14
```

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 precomputed distance matrix either from a previous call to `cmp.cluster` or from other distance calculation subroutines.

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 a descriptor database, 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.

The following example uses the clustering result obtained above using cutoff values 0.65 and 0.5. By default, the `cluster.sizestat` uses the first cutoff value:

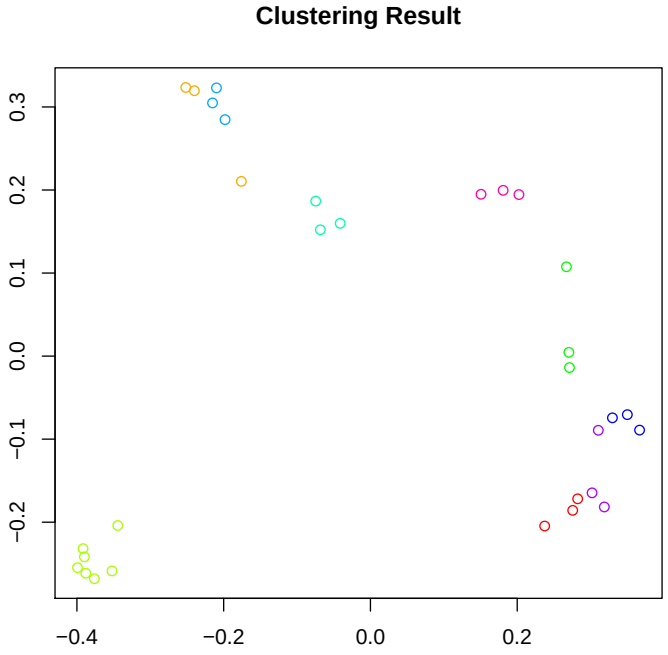
```
> cluster.sizestat(clusters)

  cluster size count
1            1    90
2            2     4
3            3     8
4            7     1
```

Based on this size statistics, clusters of size 3 or larger will be included in the following MDS visualization step.

```
> coord <- cluster.visualize(db, clusters, 3, quiet = TRUE)

============================================================
| Click points in a plot to get information on compounds    |
| they represent.                                           |
|                                                           |
| press ESC key in the plot window to stop.                 |
============================================================
```



By default `cluster.visualize` will draw the scatter plot in the R plotting device, and the user can interactively click a point to retrieve more information on the corresponding compounds. In the non-interactive mode (`non.interactive`), it will save the plot to a specified file in EPS or PDF format.

A 3D MDS plot can be created with the following sequence of commands.

```
> coord <- cluster.visualize(db, clusters, 3, dimensions = 3,
+     quiet = TRUE)
> library(scatterplot3d)
> scatterplot3d(coord)
```

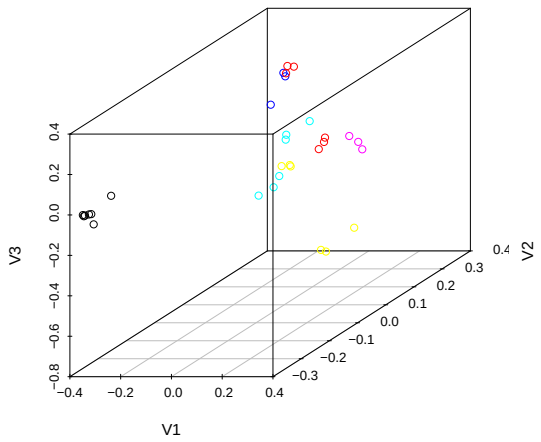


Figure 4: 3D MDS plot generated using coordinate information returned by `cluster.visualize` and the *scatterplot3d* package.

The data returned by the `cluster.visualize` can also be inspected with the fully interactive *rggobi* data visulalization system. The GGobi software and its dependencies can be obtained from the GGobi project site (`http://www.ggobi.org/rggobi`). The following commands demonstrate the import of the generated MDS data set into *rggobi*.

```
> library(rggobi)
> ggobi(coord)
```

Clustering with Other Packages

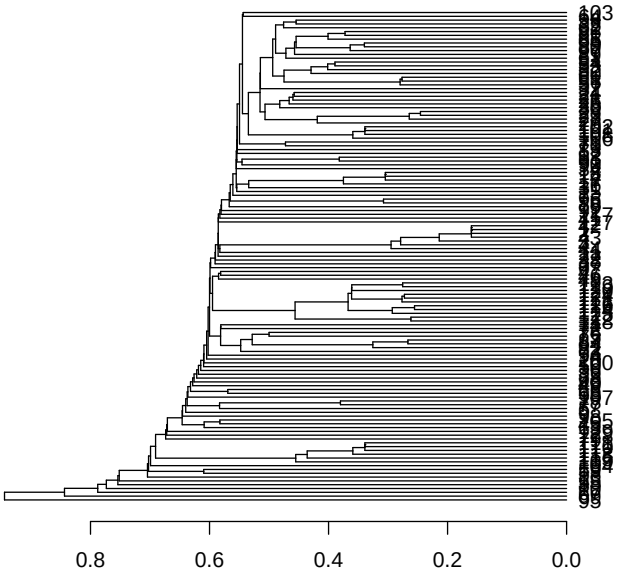
ChemmineR allows the user to take advantage of the wide spectrum of clustering utilities available in R. An example on how to perform hierarchical clustering with the `hclust` function is given below.

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

sorting result...
```

The `cmp.cluster` function is used with the `save.distances="distmat.rda"` argument to generate a distance matrix. The matrix is saved to a file named `'distmat.rda'` and it needs to be loaded into R with the `load` function. This matrix can be directly passed on to `hclust`.

```
> load("distmat.rda")
> hc <- hclust(as.dist(distmat), "single")
> plot(as.dendrogram(hc), horiz = T)
```



Format Interconversions between SMILES and SDF

This option will be provided in the future. At this point, SMILES strings can be imported into ChemmineR only indirectly by converting them into SDFs via ChemMine's online WorkBench (`http://bioweb.ucr.edu/ChemMineV2/work/smiles/`).

Calculation of Physicochemical Descriptors

Several functions will be available in the near future for calculating physicochemical descriptors directly in ChemmineR. Currently, users can calculate 40 common physicochemical descriptors with the online descriptor prediction tool available on ChemMine's WorkBench (`http://bioweb.ucr.edu/ChemMineV2/work/sdf/`).

References

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.

T Girke, L C Cheng, and N Raikhel. ChemMine. A compound mining database for chemical genomics. *Plant Physiol*, 138(2):573–577, 2005. doi: 10.1104/pp.105.062687.

Rajarshi Guha. Chemical informatics functionality in R. *Journal of Statistical Software*, 18(5), 2007.