fmcsR: a Flexible Maximum Common Substructure Algorithm for Advanced Compound Similarity Searching

Yan Wang, Tyler Backman, Kevin Horan, Thomas Girke

August 27, 2013

1 Introduction

Maximum common substructure (MCS) algorithms rank among the most sensitive and accurate methods for measuring structural similarities among small molecules. This utility is critical for many research areas in drug discovery and chemical genomics. The MCS problem is a graph-based similarity concept that is defined as the largest substructure (sub-graph) shared among two compounds (Cao et al., 2008b; Wang et al., 2013). It fundamentally differs from the structural descriptor-based strategies like fingerprints or structural keys. Another strength of the MCS approach is the identification of the actual MCS that can be mapped back to the source compounds in order to pinpoint the common and unique features in their structures. This output is often more intuitive to interpret and chemically more meaningful than the purely numeric information returned by descriptor-based approaches. Because the MCS problem is NP-complete, an efficient algorithm is essential to minimize the compute time of its extremely complex search process. The *fmcsR* package implements an efficient backtracking algorithm that introduces a new flexible MCS (FMCS) matching strategy to identify MCSs among compounds containing atom and/or bond mismatches. In contrast to this, other MCS algorithms find only exact MCSs that are perfectly contained in two molecules. The details about the FMCS algorithm are described in the Supplementary Materials Section of the associated publication (Wang et al., 2013). The package provides several utilities to use the FMCS algorithm for pairwise compound comparisons, structure similarity searching and clustering. To maximize performance, the time consuming computational steps of *fmcsR* are implemented in C++. Integration with the *ChemmineR* package provides visualization functionalities of MCSs and consistent structure and substructure data handling routines (Cao et al., 2008a; Backman et al., 2011). The following gives an overview of the most important functionalities provided by *fmcsR*.

2 Installation

The R software for running *fmcsR* and *ChemmineR* can be downloaded from CRAN (`http://cran.at.r-project.org/`). The *fmcsR* package can be installed from an open R session using the `biocLite` install command.

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("fmcsR")
```

3 Quick Overview

To demo the main functionality of the *fmcsR* package, one can load its sample data stored as *SDFset* object. The generic `plot` function can be used to visualize the corresponding structures.

```
> library(fmcsR)
> data(fmcstest)
> plot(fmcstest[1:3], print=FALSE)
```

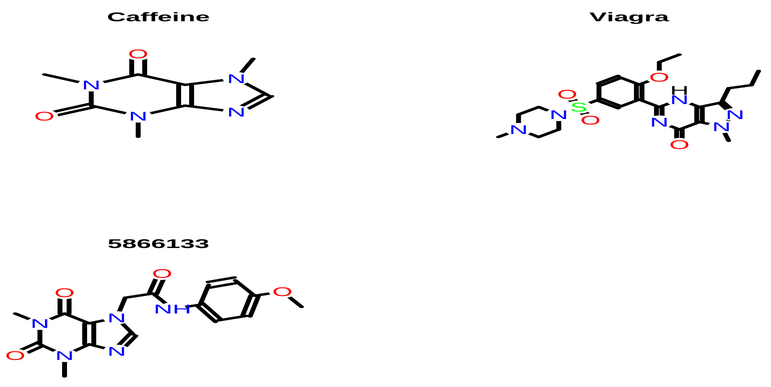


Figure 1: Structures depictions of sample data.

The `fmcs` function computes the MCS/FMCS shared among two compounds, which can be highlighted in their structure with the `plotMCS` function.

```
> test <- fmcs(fmcstest[1], fmcstest[2], au=2, bu=1)
> plotMCS(test)
```

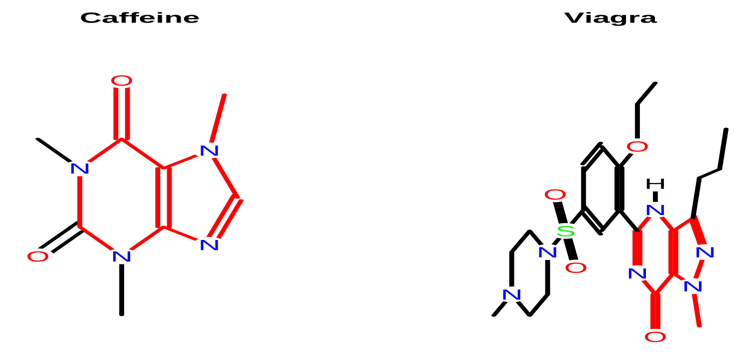


Figure 2: The red bonds highlight the MCS shared among the two compounds.

4 Documentation

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

```
> library(help="fmcsR") # Lists functions/classes provided by fmcsR
> library(help="ChemmineR") # Lists functions/classes from ChemmineR
> vignette("fmcsR") # Opens this PDF manual
> vignette("ChemmineR") # Opens ChemmineR PDF manual
```

The help documents for the different functions and container classes can be accessed with the standard R help syntax.

```
> ?fmcs
> ?"MCS-class"
> ?"SDFset-class"
```

5 MCS of Two Compounds

5.1 Data Import

The following loads the sample data set provided by the *fmcsR* package. It contains the SD file (SDF) of 3 molecules stored in an *SDFset* object.

```
> data(fmcstest)
> sdfset <- fmcstest
> sdfset

An instance of "SDFset" with 3 molecules
```

Custom compound data sets can be imported and exported with the `read.SDFset` and `write.SDF` functions, respectively. The following demonstrates this by exporting the *sdfset* object to a file named sdfset.sdf. The latter is then reimported into R with the `read.SDFset` function.

```
> write.SDF(sdfset, file="sdfset.sdf")
> mysdf <- read.SDFset(file="sdfset.sdf")
```

5.2 Compute MCS

The `fmcs` function accepts as input two molecules provided as *SDF* or *SDFset* objects. Its output is an S4 object of class *MCS*. The default printing behavior summarizes the MCS result by providing the number of MCSs it found, the total number of atoms in the query compound a, the total number of atoms in the target compound b, the number of atoms in their MCS c and the corresponding *Tanimoto Coefficient*. The latter is a widely used similarity measure that is defined here as $c/(a + b - c)$. In addition, the *Overlap Coefficient* is provided, which is defined as $c/min(a, b)$. This coefficient is often useful for detecting similarities among compounds with large size differences.

```
> mcsa <- fmcs(sdfset[[1]], sdfset[[2]])
> mcsa

An instance of "MCS"
 Number of MCSs: 7
 CMP1: 14 atoms
 CMP2: 33 atoms
 MCS: 8 atoms
 Tanimoto Coefficient: 0.20513
 Overlap Coefficient: 0.57143

> mcsb <- fmcs(sdfset[[1]], sdfset[[3]])
> mcsb

An instance of "MCS"
 Number of MCSs: 1
 CMP1: 14 atoms
 CMP2: 25 atoms
 MCS: 14 atoms
 Tanimoto Coefficient: 0.56
 Overlap Coefficient: 1
```

If `fmcs` is run with `fast=TRUE` then it returns the numeric summary information in a named *vector*.

```
> fmcs(sdfset[1], sdfset[2], fast=TRUE)
```

```
          Query_Size          Target_Size             MCS_Size 
          14.0000000           33.0000000            8.0000000 
Tanimoto_Coefficient  Overlap_Coefficient 
           0.2051282            0.5714286 
```

5.3 *MCS* Class Usage

The *MCS* class contains three components named *stats*, *mcs1* and *mcs2*. The *stats* slot stores the numeric summary information, while the structural MCS information for the query and target structures is stored in the *mcs1* and *mcs2* slots, respectively. The latter two slots each contain a *list* with two subcomponents: the original query/target structures as *SDFset* objects as well as one or more numeric index vector(s) specifying the MCS information in form of the row positions in the atom block of the corresponding *SDFset*. A call to `fmcs` will often return several index vectors. In those cases the algorithm has identified alternative MCSs of equal size.

```
> slotNames(mcsa)
```

```
[1] "stats" "mcs1"  "mcs2"
```

Accessor methods are provided to return the different data components of the *MCS* class.

```
> stats(mcsa) # or mcsa[["stats"]]
```

```
          Query_Size          Target_Size             MCS_Size 
          14.0000000           33.0000000            8.0000000 
Tanimoto_Coefficient  Overlap_Coefficient 
           0.2051282            0.5714286 
```

```
> mcsa1 <- mcs1(mcsa) # or mcsa[["mcs1"]]
> mcsa2 <- mcs2(mcsa) # or mcsa[["mcs2"]]
> mcsa1[1] # returns SDFset component
```

```
$query
An instance of "SDFset" with 1 molecules
```

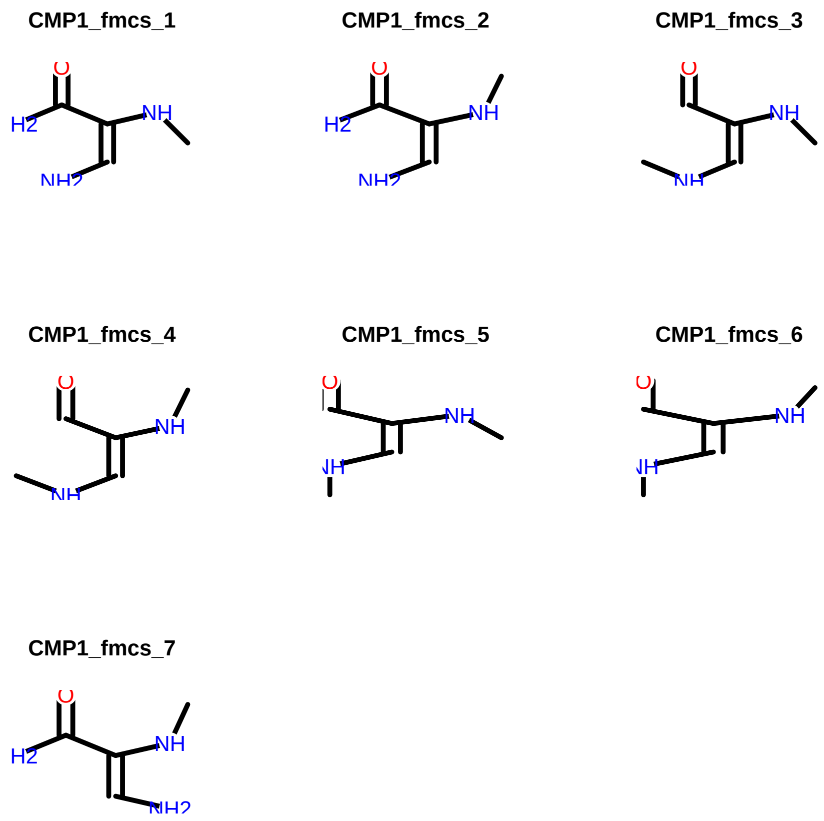
```
> mcsa1[[2]][1:2] # return first two index vectors
```

```
$CMP1_fmcs_1
[1]  3  8  7  4  9  5 11  1

$CMP1_fmcs_2
[1]  3  8  7  4  9  5  1 13
```

The `mcs2sdfset` function can be used to return the substructures stored in an *MCS* instance as *SDFset* object. If `type="new"` new atom numbers will be assigned to the subsetted SDF, while `type="old"` will maintain the atom numbers from its source. For details consult the help documents *?mcs2sdfset* and *?atomsubset*.

```
> mcstosdfset <- mcs2sdfset(mcsa, type="new")
> plot(mcstosdfset[[1]], print=FALSE)
```



To construct an *MCS* object manually, one can provide the required data components in a *list*.

```
> mylist <- list(stats=stats(mcsa), mcs1=mcs1(mcsa), mcs2=mcs2(mcsa))
> as(mylist, "MCS")
```

```
An instance of "MCS"
 Number of MCSs: 7
 CMP1: 14 atoms
 CMP2: 33 atoms
 MCS: 8 atoms
 Tanimoto Coefficient: 0.20513
 Overlap Coefficient: 0.57143
```

6 FMCS of Two Compounds

If `fmcs` is run with its default paramenters then it returns the MCS of two compounds, because the mismatch parameters are all set to zero. To identify FMCSs, one has to raise the number of upper bound atom mismates `au` and/or bond mismatches `bu` to interger values above zero.

```
> plotMCS(fmcs(sdfset[1], sdfset[2], au=0, bu=0))
```

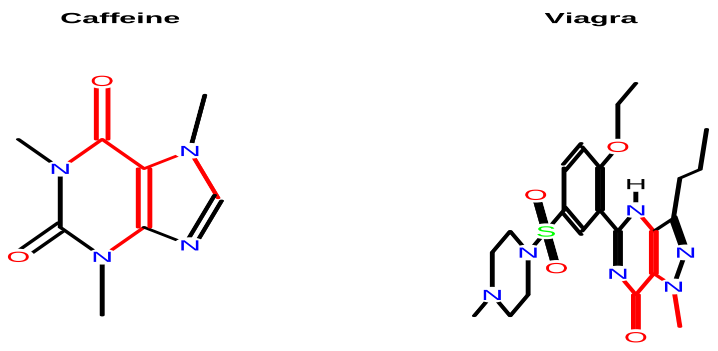


Figure 3: MCS for *sdfset[1]* and *sdfset[2]* with `au=0` and `bu=0`

```
> plotMCS(fmcs(sdfset[1], sdfset[2], au=1, bu=1))
```

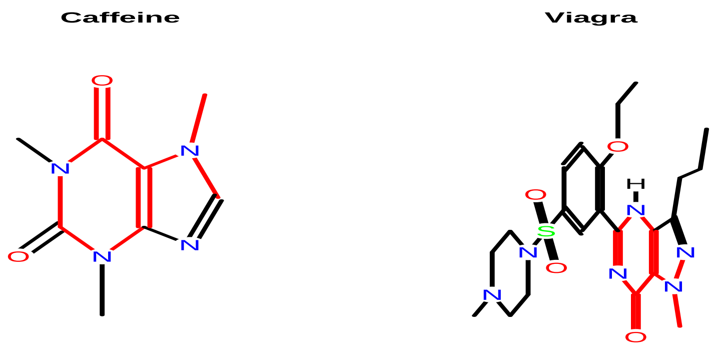


Figure 4: FMCS for *sdfset[1]* and *sdfset[2]* with `au=1` and `bu=1`

```
> plotMCS(fmcs(sdfset[1], sdfset[2], au=2, bu=2))
```

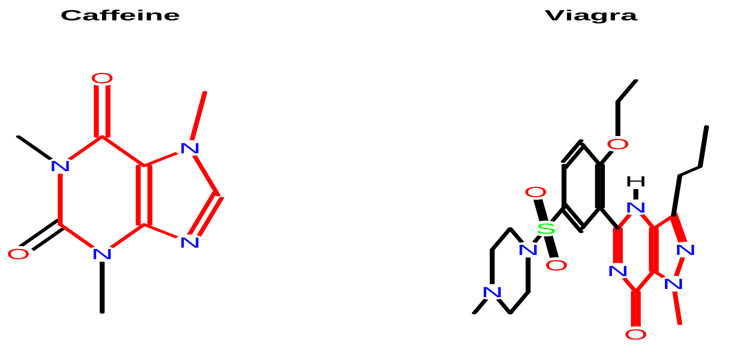


Figure 5: FMCS for *sdfset[1]* and *sdfset[2]* with `au=2` and `bu=2`

```
> plotMCS(fmcs(sdfset[1], sdfset[3], au=0, bu=0))
```

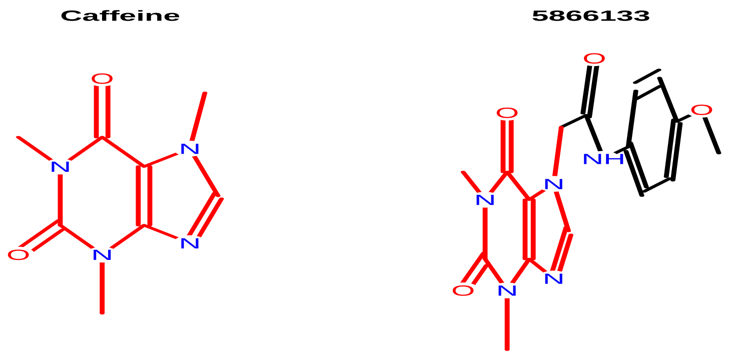


Figure 6: MCS for *sdfset[1]* and *sdfset[3]* with `au=0` and `bu=0`

7 FMCS Search Functionality

The `fmcsBatch` function provides FMCS search functionality for compound collections stored in *SDFset* objects.

```
> data(sdfsample) # Loads larger sample data set
> sdf <- sdfsample
> fmcsBatch(sdf[1], sdf[1:30], au=0, bu=0)
```

```
      Query_Size Target_Size MCS_Size Tanimoto_Coefficient Overlap_Coefficient
CMP1          33          33       33            1.0000000           1.0000000
CMP2          33          26       11            0.2291667           0.4230769
CMP3          33          26       10            0.2040816           0.3846154
CMP4          33          32        9            0.1607143           0.2812500
CMP5          33          23       14            0.3333333           0.6086957
CMP6          33          19       13            0.3333333           0.6842105
CMP7          33          21        9            0.2000000           0.4285714
CMP8          33          31        8            0.1428571           0.2580645
CMP9          33          21        9            0.2000000           0.4285714
CMP10         33          21        8            0.1739130           0.3809524
CMP11         33          36       15            0.2777778           0.4545455
CMP12         33          26       12            0.2553191           0.4615385
CMP13         33          26       11            0.2291667           0.4230769
CMP14         33          16       12            0.3243243           0.7500000
CMP15         33          34       15            0.2884615           0.4545455
CMP16         33          25        8            0.1600000           0.3200000
CMP17         33          19        8            0.1818182           0.4210526
CMP18         33          24       10            0.2127660           0.4166667
CMP19         33          25       14            0.3181818           0.5600000
CMP20         33          26       10            0.2040816           0.3846154
CMP21         33          25       15            0.3488372           0.6000000
CMP22         33          21       11            0.2558140           0.5238095
CMP23         33          26       11            0.2291667           0.4230769
CMP24         33          17        6            0.1363636           0.3529412
CMP25         33          27        9            0.1764706           0.3333333
CMP26         33          24       13            0.2954545           0.5416667
CMP27         33          26       11            0.2291667           0.4230769
CMP28         33          20       10            0.2325581           0.5000000
CMP29         33          20        8            0.1777778           0.4000000
CMP30         33          18        7            0.1590909           0.3888889
```

8 Clustering with FMCS

The `fmcsBatch` function can be used to compute a similarity matrix for clustering with various algorithms available in R. The following example uses the FMCS algorithm to compute a similarity matrix that is used for hierarchical clustering with the `hclust` function and the result is plotted in form of a dendrogram.

```
> sdf <- sdf[1:7]
> d <- sapply(cid(sdf), function(x)
+             fmcsBatch(sdf[x], sdf, au=0, bu=0,
+             matching.mode="aromatic")[,"Overlap_Coefficient"])
> d
```

```
          CMP1      CMP2      CMP3      CMP4      CMP5      CMP6      CMP7
CMP1 1.0000000 0.2307692 0.2307692 0.2812500 0.5217391 0.6842105 0.2857143
CMP2 0.2307692 1.0000000 0.4230769 0.5384615 0.2173913 0.4736842 0.2857143
CMP3 0.2307692 0.4230769 1.0000000 0.3076923 0.2173913 0.4736842 0.9047619
CMP4 0.2812500 0.5384615 0.3076923 1.0000000 0.3043478 0.5263158 0.2857143
CMP5 0.5217391 0.2173913 0.2173913 0.3043478 1.0000000 0.5789474 0.2380952
CMP6 0.6842105 0.4736842 0.4736842 0.5263158 0.5789474 1.0000000 0.3157895
CMP7 0.2857143 0.2857143 0.9047619 0.2857143 0.2380952 0.3157895 1.0000000
```

```
> hc <- hclust(as.dist(1-d), method="complete")
> plot(as.dendrogram(hc), edgePar=list(col=4, lwd=2), horiz=TRUE)
```

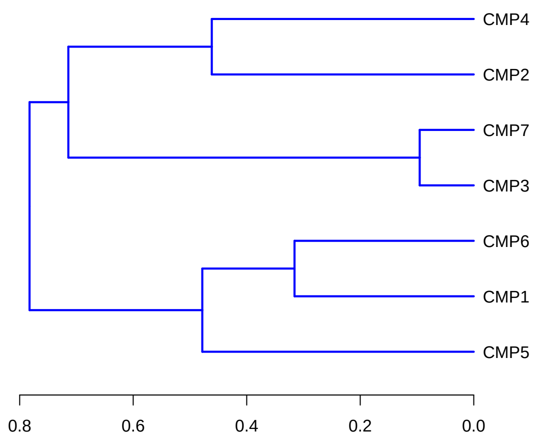


Figure 7: Hierarchical clustering result.

The FMCS shared among compound pairs of interest can be visualized with `plotMCS`, here for the two most similar compounds from the previous tree:

```
> plotMCS(fmcs(sdf[3], sdf[7], au=0, bu=0, matching.mode="aromatic"))
```

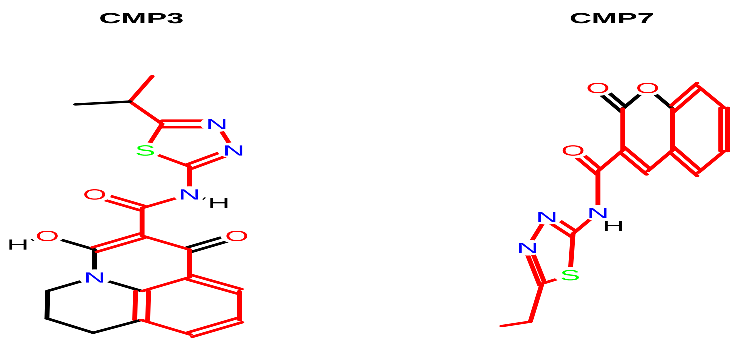


Figure 8: Most similar compounds from previous tree.

9 Version Information

```
> sessionInfo()

R version 3.0.1 (2013-05-16)
Platform: x86_64-unknown-linux-gnu (64-bit)

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=C                 LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

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

other attached packages:
[1] fmcsR_1.2.1      ChemmineR_2.12.2

loaded via a namespace (and not attached):
[1] DBI_0.2-7      RCurl_1.95-4.1 digest_0.6.3   tools_3.0.1
```

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 2008a. doi: 10.1093/bioinformatics/btn307. URL `http://www.hubmed.org/display.cgi?uids=18596077`.

Y Cao, T Jiang, and T Girke. A maximum common substructure-based algorithm for searching and predicting drug-like compounds. *Bioinformatics*, 24(13):366–374, Jul 2008b. doi: 10.1093/bioinformatics/btn186. URL `http://www.hubmed.org/display.cgi?uids=18586736`.

Y Wang, T W Backman, K Horan, and T Girke. fmcsR: Mismatch Tolerant Maximum Common Substructure Searching in R. *Bioinformatics*, Aug 2013. doi: 10.1093/bioinformatics/btt475. URL `http://www.hubmed.org/display.cgi?uids=23962615`.