

# Package ‘SFS’

January 20, 2025

**Type** Package

**Title** Similarity-First Search Seriation Algorithm

**Version** 0.1.4

**Date** 2019-05-06

**Maintainer** Utz-Uwe Haus <uhaus@cray.com>

**Description** An implementation of the Similarity-First Search algorithm (SFS), a combinatorial algorithm which can be used to solve the seriation problem and to recognize some structured weighted graphs. The SFS algorithm represents a generalization to weighted graphs of the graph search algorithm Lexicographic Breadth-First Search (Lex-BFS), a variant of Breadth-First Search. The SFS algorithm reduces to Lex-BFS when applied to binary matrices (or, equivalently, unweighted graphs). Hence this library can be also considered for Lex-BFS applications such as recognition of graph classes like chordal or unit interval graphs. In fact, the SFS seriation algorithm implemented in this package is a multisweep algorithm, which consists in repeating a finite number of SFS iterations (at most  $n$  sweeps for a matrix of size  $n$ ). If the data matrix has a Robinsonian structure, then the ranking returned by the multistep SFS algorithm is a Robinson ordering of the input matrix. Otherwise the algorithm can be used as a heuristic to return a ranking partially satisfying the Robinson property.

**License** GPL-3

**Encoding** UTF-8

**Imports** Rcpp (>= 0.12.7)

**Suggests** seriation

**LinkingTo** Rcpp, RcppArmadillo

**SystemRequirements** C++11

**RoxygenNote** 6.1.1

**NeedsCompilation** yes

**Author** Matteo Seminaroti [aut, cph],  
Utz-Uwe Haus [aut, cre, cph],  
Monique Laurent [ctb]

**Repository** CRAN

**Date/Publication** 2019-05-07 07:30:04 UTC

## Contents

SFS_read	2
SFS_sfs	3
<b>Index</b>	<b>6</b>

---

SFS_read	<i>Read similarity or dissimilarity input data</i>
----------	--

---

### Description

Read the similarity (or dissimilarity) information between the objects that one wants to order and build a 3-columns data frame, where each row  $(i, j, A_{ij})$  represents the (dis)similarity  $A_{ij}$  between objects  $i$  and  $j$ . In case of symmetric data (i.e.,  $A_{ij} = A_{ji}$ ), only the entries for pairs  $(i, j)$  with  $i < j$  are listed.

### Usage

```
read(data, zero_epsilon = 1e-200, symmetric = TRUE, identical_val = FALSE)
```

### Arguments

<code>data</code>	a representation of the similarity (or dissimilarity) between pairs of objects.
<code>zero_epsilon</code>	a numerical value which determines that values in data below this threshold are considered to be $\emptyset$ .
<code>symmetric</code>	a boolean value equal to TRUE if the input data is a symmetric matrix (i.e., $A_{ij} = A_{ji}$ for all $i$ and $j$ ).
<code>identical_val</code>	a boolean value equal to TRUE if the data is given as a 3-columns data frame and entries at both positions $(i, j)$ and $(j, i)$ are included.

### Details

The input data can be a weighted adjacency matrix (represented by the objects: matrix, dist or data frame), or a list of all the weighted edges of a weighted graph (represented by a 3-col data frame) where each row  $(i, j, A_{ij})$  represents the (dis)similarity  $A_{ij}$  between objects  $i$  and  $j$  with  $i < j$ . If not specified, the data is assumed to be symmetric (i.e., same entry at positions  $(i, j)$  and  $(j, i)$ ). Since by default the data is assumed to be symmetric, if it is represented by a 3-columns data frame, then it is assumed that symmetric pairs are not listed, and thus by default `identical_val = FALSE`. The reason for this choice is that for large symmetric data, it is more efficient to list the symmetric entries only once. However, note that if `symmetric = FALSE` then `identical_val = TRUE` automatically.

### Value

Returns a 3-columns data frame representation of the original data listing all the pairwise (dis)similarities  $(i, j, A_{ij})$  between objects and selecting only the entries  $A_{ij}$  with  $i < j$  when the data is a symmetric matrix  $A$ .

**Author(s)**

Matteo Seminaroti (SFS) and Utz-Uwe Haus (R wrapping)

---

SFS\_sfs

*Similarity-First Search multisweep algorithm*


---

**Description**

Return a ranking of the objects such that similar objects are ordered close to each other. If the matrix is *Robinsonian*, then the ranking returned is a *Robinson ordering*.

**Usage**

```
sfs(matrix, sfs_epsilon = 0, dissimilarity = FALSE, Robinsonian = FALSE, num_sweeps = 4)
```

**Arguments**

<code>matrix</code>	a 3-columns data frame with no repeated symmetric entries, representing the list of all similarities (or dissimilarities) $(i, j, A_{ij})$ between the pairs of objects to reorder.
<code>sfs_epsilon</code>	a numerical value which determines that two entries whose difference is below this threshold are considered to be equal.
<code>dissimilarity</code>	a boolean value equal to TRUE if the input data is a dissimilarity.
<code>Robinsonian</code>	a boolean value equal to TRUE if one wants to recognize a Robinsonian matrix.
<code>num_sweeps</code>	an integer value that determines how many iterations of SFS shall be performed.

**Details**

Given a 3-columns data frame  $(i, j, A_{ij})$  listing all the similarities (or dissimilarities) among the objects, this function builds a `spMat` object in *Armadillo* and computes a finite number of repeated SFS iterations (each called a *sweep*). The user may decide the threshold for which two entries are considered equal, meaning that if  $|A_{ij} - A_{ik}| \leq \text{sfs\_epsilon}$ , then objects  $j$  and  $k$  have the same similarity (or dissimilarity) with respect to object  $i$ . By default, this threshold is set to 0.

If not specified, the `matrix` represents the similarity information between objects. If `dissimilarity = TRUE`, then the `matrix` represents the dissimilarity information and the SFS algorithm is modified by sorting the neighborhood of a visited vertex for increasing values (instead of for decreasing values).

The parameter `k = num_sweeps` sets the number of sweeps performed by `SFS()`. This number directly affects the complexity of the function since, as each sweep runs in  $(k(n + m \log n))$  time, `SFS()` runs in  $(k(n + m \log n))$  time. By default, `num_sweeps=4`, as it is known that three sweeps suffice for recognizing Robinsonian binary matrices and for general matrices experiments show that four sweeps are enough for finding a good ranking for most data. If `Robinsonian = TRUE`, then the number of sweeps is automatically set to the number of objects  $n$  to rank minus one. In this case, `sfs()` also checks if the returned permutation is a Robinson ordering (since it is known that if the order returned after  $n - 1$  sweeps is not a Robinson ordering then the data is not Robinsonian). Efficient measures are implemented in order to avoid unnecessary time consuming loops between

consecutive SFS iterations. Note that checking if a given permutation is a Robinson ordering is implemented at the moment only when dealing with similarities among the objects.

Finally, the object returned by `SFS()` is a vector of integers, where the entry at position  $i$  represents the  $i$ -th object in the ranking. If the matrix is 0-based, then the returned vector is 0-based. If matrix is 1-based, then the returned vector is 1-based.

### Value

Return a (row) vector of integers representing the ranking of the objects, which is 0-based or 1-based accordingly with the input matrix.

### Author(s)

Matteo Seminaroti (SFS) and Utz-Uwe Haus (R wrapping)

### References

M. Laurent and M. Seminaroti. Similarity-First Search: a new algorithm with application to Robinsonian matrix recognition. *SIAM Journal on Discrete Mathematics* (to appear). *arXiv:1601.03521*. 2016.

M. Seminaroti. Combinatorial Algorithms for the Seriation Problem. *PhD thesis*. School of Economics and Management, Tilburg University, pages 1–209. 2016.

### Examples

```
## install package in R
#install.packages("SFS_0.1.tar.gz")
#install.packages("seriation")

## load package in R
library(SFS)

## invoke SFS on a R Matrix
mtxt <- c("11 2 9 0 5 0 5 5 2 0 5 0 5 6 0 0 2 0 5",
          "2 11 2 0 9 0 8 5 10 0 5 0 5 2 0 0 10 0 8",
          "9 2 11 0 5 0 5 5 2 0 5 0 5 10 0 0 2 0 5",
          "0 0 0 11 0 3 0 0 0 3 0 3 0 0 10 3 0 9 0",
          "5 9 5 0 11 0 8 7 9 0 7 0 7 5 0 0 9 0 10",
          "0 0 0 3 0 11 0 0 0 10 0 6 0 0 5 8 0 5 0",
          "5 8 5 0 8 0 11 7 8 0 7 0 7 5 0 0 8 0 9",
          "5 5 5 0 7 0 7 11 6 0 10 0 8 7 0 0 6 0 7",
          "2 10 2 0 9 0 8 6 11 0 6 0 5 2 0 0 10 0 8",
          "0 0 0 3 0 10 0 0 0 11 0 6 0 0 4 9 0 5 0",
          "5 5 5 0 7 0 7 10 6 0 11 0 9 7 0 0 6 0 7",
          "0 0 0 3 0 6 0 0 0 6 0 11 0 0 9 6 0 10 0",
          "5 5 5 0 7 0 7 8 5 0 9 0 11 7 0 0 5 0 7",
          "6 2 10 0 5 0 5 7 2 0 7 0 7 11 0 0 2 0 5",
          "0 0 0 10 0 5 0 0 0 4 0 9 0 0 11 4 0 10 0",
          "0 0 0 3 0 8 0 0 0 9 0 6 0 0 4 11 0 4 0",
          "2 10 2 0 9 0 8 6 10 0 6 0 5 2 0 0 11 0 8",
          "0 0 0 9 0 5 0 0 0 5 0 10 0 0 10 4 0 11 0")
```

```

      "5 8 5 0 10 0 9 7 8 0 7 0 7 5 0 0 8 0 11")
M <- as.matrix(read.table(textConnection(mtxt)))
A <- SFS::read(M)
SFS::sfs(A, Robinsonian = TRUE)

## invoke SFS on a data-frame with 3-column data-frames with 0-based vertices, with
## (row, col, value) triples containing symmetric values
data <- c("0 0 1.0",
         "1 0 1.5",
         "2 0 1.9",
         "0 1 2.0",
         "1 1 2.5",
         "2 1 2.9",
         "0 2 3.0",
         "1 2 3.5",
         "2 2 3.9")
M <- read.table(textConnection(data))
A <- SFS::read(M, identical_val = TRUE)
SFS::sfs(A)

## invoke SFS on a \code{dist} from seriation package:
library(seriation)
data("iris");
x <- as.matrix(iris[-5]);
x <- x[sample(1:nrow(x)),];
M <- dist(x)
D <- SFS::read(M)
SFS::sfs(D, sfs_epsilon = 0.01, dissimilarity = TRUE)

## invoke SFS reading from file a Robinsonian matrix
M <- read.table(system.file("extdata", "list_130.txt", package = "SFS"))
A <- SFS::read(M, identical_val = TRUE)
SFS::sfs(A, Robinsonian = TRUE)

## invoke SFS reading from file containing 3-columns (row, col, value) entries
## of an asymmetric non-Robinsonian similarity matrix with 1-based vertices
M <- read.table(system.file("extdata", "list_130.txt", package = "SFS"))
A <- SFS::read(M, identical_val = TRUE, symmetric = FALSE)
SFS::sfs(A, num_sweeps = 7)

```

# Index

read (SFS\_read), 2

sfs (SFS\_sfs), 3

SFS\_read, 2

SFS\_sfs, 3