Clustering in an Object-Oriented Environment

Anja Struyf* Mia Hubert Peter J. Rousseeuw

Department of Mathematics and Computer Science, U.I.A.,
Universiteitsplein 1, B-2610 Antwerp, Belgium

Abstract

This paper describes the incorporation of seven stand-alone clustering programs into S-PLUS, where they can now be used in a much more flexible way. The original Fortran programs carried out new cluster analysis algorithms introduced in the book of Kaufman and Rousseeuw (1990). These clustering methods were designed to be robust and to accept dissimilarity data as well as objects-by-variables data. Moreover, they each provide a graphical display and a quality index reflecting the strength of the clustering. The powerful graphics of S-PLUS made it possible to improve these graphical representations considerably. The integration of the clustering algorithms was performed according to the object-oriented principle supported by S-PLUS. The new functions have a uniform interface, and are compatible with existing S-PLUS functions. We will describe the basic idea and the use of each clustering method, together with its graphical features. Each function is briefly illustrated with an example.

Keywords: Fuzzy clustering; Graphics; Medoids; Object-oriented programming; Statistical software.

1 Introduction

The book *Finding Groups in Data* (Kaufman and Rousseeuw 1990, hereafter [KR]) describes several modern techniques for cluster analysis, together with their motivation, algorithms,

*supported by the Belgian Science Foundation (NFWO).

and examples. These techniques were originally implemented as stand-alone Fortran programs by the names of DAISY, PAM, CLARA, FANNY, AGNES, DIANA and MONA, collectively known as the library CLUSFIND.

The purpose of our work was to incorporate these methods into S-PLUS (1993). For this the Fortran routines needed to be made compatible with S-PLUS conventions, and inside S-PLUS interfaces (functions) had to be constructed to make use of the Fortran code in an optimal way. Many elementary operations (such as checking the input data and performing initial data transformations) were moved from the Fortran code to S-PLUS, which of course has all the necessary tools already available. Moreover, S-PLUS offers excellent graphical capabilities, which allowed us to shorten those portions of the Fortran code substantially. Naturally, it has been checked throughout that the resulting S-PLUS functions yield exactly the same result as the original programs. Moreover, help files for all the new functions and objects have been prepared. The new functions are easy to use in the object-oriented environment of S-PLUS.

Generally speaking, cluster analysis methods are of either of two types:

Partitioning methods: algorithms that divide the dataset into k clusters, where the integer k needs to be specified by the user. Typically, the user runs the algorithm for a range of k-values. For each k, the algorithm carries out the clustering and also yields a "quality index", which allows the user to select a value of k afterwards.

Hierarchical methods: algorithms yielding an entire hierarchy of clusterings of the dataset. *Agglomerative* methods start with the situation where each object in the dataset forms its own little cluster, and then successively merge clusters until only one large cluster remains which is the whole dataset. *Divisive* methods start by considering the whole dataset as one cluster, and then split up clusters until each object is separate.

Three of the algorithms considered here are of the partitioning type: `pam`, `clara`, and `fanny`. The algorithms `agnes`, `diana`, and `mona` are of the hierarchical type.

Datasets for clustering can have either of the following structures:

- an $n \times p$ objects-by-attributes matrix, where rows stand for objects and columns stand for variables;
- an $n \times n$ dissimilarity matrix, where $d(i, j) = d(j, i)$ measures the "difference" or

dissimilarity between the objects i and j. This kind of data occurs frequently in the social sciences and in marketing.

Many existing clustering methods can only process an $n \times p$ objects-by-attributes matrix, whereas most methods considered here can operate on a dissimilarity matrix. (If the data happens to consist of an $n \times p$ data matrix, these methods first construct the corresponding dissimilarity matrix.) To save storage space, S-PLUS represents a dissimilarity matrix as a vector rather than a full matrix. Only the upper-triangular half of the symmetric matrix is considered, and placed rowwise into a vector. The Fortran programs use the lower-triangular dissimilarity matrix instead, hence the new S-PLUS functions transform the dissimilarity vector internally.

Apart from the 6 clustering algorithms mentioned above, [KR, chapter 1] also provide the auxiliary program DAISY which computes a dissimilarity matrix from objects-by-attributes. This program has been incorporated as the S-PLUS function

```
daisy(x, metric = "euclidean", stand = F, type = list()).
```

One argument is always required when calling this function:

- `x`: a matrix or dataframe, containing the measurements. Columns of class `numeric` will here be recognized as interval scaled variables, columns of class `factor` as nominal, and of class `ordered` as ordinal variables. Other variable types should be specified with the `type` argument.

The other arguments are optional:

- `metric`: when all variables are numeric (i.e. continuous on a linear scale) the `metric` argument specifies whether dissimilarities will be computed using the `euclidean` or the `manhattan` metric.

- `stand`: measurements will be standardized to zero location and unit spread with the option `stand=T`. This option is only used when all variables are numeric. Standardization is often useful to avoid depending on the choice of measurement units. In S-PLUS standardization is easily performed with the existing `scale` function, which allowed us to remove this part of the computation from the Fortran code.

- `type`: this argument is used to specify the type of the variables which are not automatically recognized by S-PLUS. Possible types are `ordratio`, `logratio`, and `asymm`. (See also Example 2.)

Compared to the existing S-PLUS function `dist` whose input must be numeric or binary variables, the main feature of `daisy` is its ability to handle also nominal, ordinal, asymmetric binary and ratio-scaled variables, even if different types of variables occur in the same dataset. The combination of variables of mixed type into a single dissimilarity matrix slightly extends a definition of Gower (1971), by covering also ordinal and ratio variables. This results in the dissimilarity $d(i,j)$ defined as

$$d(i,j) = \frac{\sum_{f=1}^{p} \delta_{ij}^{(f)} d_{ij}^{(f)}}{\sum_{f=1}^{p} \delta_{ij}^{(f)}} \qquad \in [0,1]$$

where

$d_{ij}^{(f)}$ = contribution of variable f to $d(i,j)$, which depends on its type:

- f binary or nominal: $d_{ij}^{(f)} = 0$ if $x_{if} = x_{jf}$, and $d_{ij}^{(f)} = 1$ otherwise,
- f interval-scaled: $d_{ij}^{(f)} = \frac{|x_{if} - x_{jf}|}{\max_h x_{hf} - \min_h x_{hf}}$,
- f ordinal or ratio-scaled: compute ranks r_{if} and $z_{if} = \frac{r_{if}-1}{\max_h r_{hf}-1}$ and treat these z_{if} as interval-scaled,

and

$\delta_{ij}^{(f)}$ = weight of variable f:

- $\delta_{ij}^{(f)} = 0$ if x_{if} or x_{jf} is missing,
- $\delta_{ij}^{(f)} = 0$ if $x_{if} = x_{jf} = 0$ and variable f is asymmetric binary,
- $\delta_{ij}^{(f)} = 1$ otherwise.

The output from `daisy` is an object of the class `dissimilarity`, and can be used as input for several of the clustering functions.

The following two examples illustrate the use of `daisy` on a numeric dataset, and on a dataset containing variables of different types.

Example 1. The agriculture dataset (see also Section 2.1) contains two interval scaled variables measured on 12 European countries. The S-PLUS call used to compute dissimilarities between these countries, using the Euclidean metric, and without standardization, is `daisy(agriculture,metric="euclidean",stand=F)`. This call may also be abbreviated to `daisy(agriculture)`, because we use the default values for `metric` and `stand` as specified in the function header.

Example 2. The dataset of Table 1 contains 8 characteristics for 18 flowers. The eight variables are:

1. Winters: binary, indicates whether the plant may be left in the garden when it freezes.
2. Shadow: binary, shows whether the plant needs to stand in the shadow.
3. Tubers: asymmetric binary, distinguishes between plants with tubers and plants that grow in any other way.
4. Color: nominal, specifies the flower's color (1=white, 2=yellow, 3= pink, 4=red, 5= blue).
5. Soil: ordinal, indicates whether the plant grows in dry (1), normal (2), or wet (3) soil.
6. Preference: ordinal, someone's preference ranking, going from 1 to 18.
7. Height: interval scaled, the plant's height in centimeters.
8. Distance: interval scaled, the distance in centimeters that should be left between the plants.

In S-PLUS, variables 1,2, and 4 must be represented as `factor` objects to make sure that `daisy` interprets them right. Columns 5 and 6 should be objects of class `ordered`, and columns 7 and 8 `numeric`. The type of variable 3 is specified with the `type` argument as follows: `daisy(flower,type=list(asymm=3))`. If also variable 1 were asymmetric binary, and variable 7 were a ratio scaled variable to be treated as ordinal, the call to `daisy` would be `daisy(flower,type=list(asymm=c(1,3),ordratio=7))`. Using `logratio` instead of `ordratio` would cause variable 7 to be first logarithmically transformed and then being interpreted as interval scaled.

The datasets used in these examples, as well as those used in Sections 2 and 3, can be found in data files accompanying this paper.

In the next section we describe the three partitioning methods `pam`, `clara` and `fanny`. In Section 3 we give an outline of the hierarchical methods `agnes`, `diana` and `mona`. Section 4 provides a schematic overview of the interplay between these new functions, and discusses the advantages and features of the underlying clustering methods. Finally, the appendix summarizes the object-oriented terminology in S-PLUS.

	variable							
object	1	2	3	4	5	6	7	8
Begonia	0	1	1	4	3	15	25	15
Broom	1	0	0	2	1	3	150	50
Camellia	0	1	0	3	3	1	150	50
Dahlia	0	0	1	4	2	16	125	50
Forget-me-not	0	1	0	5	2	2	20	15
Fuchsia	0	1	0	4	3	12	50	40
Geranium	0	0	0	4	3	13	40	20
Gladiolus	0	0	1	2	2	7	100	15
Heather	1	1	0	3	1	4	25	15
Hydrangea	1	1	0	5	2	14	100	60
Iris	1	1	1	5	3	8	45	10
Lily	1	1	1	1	2	9	90	25
Lily-of-the-valley	1	1	0	1	2	6	20	10
Peony	1	1	1	4	2	11	80	30
Pink Carnation	1	0	0	3	2	10	40	20
Red Rose	1	0	0	4	2	18	200	60
Scotch Rose	1	0	0	2	2	17	150	60
Tulip	0	0	1	2	1	5	25	10

Table 1: Flower dataset.

2 Partitioning methods

2.1 Partitioning Around Medoids: function `pam`

The function `pam` is based on the search for k *representative objects*, called *medoids*, among the objects of the dataset (Kaufman and Rousseeuw 1987). These medoids are computed such that the total dissimilarity of all objects to their nearest medoid is minimal: i.e. the goal is to find a subset $\{m_1, \ldots, m_k\} \subset \{1, \ldots, n\}$ which minimizes the objective function

$$\sum_{i=1}^{n} \min_{t=1,\ldots,k} d(i, m_t). \tag{2.1}$$

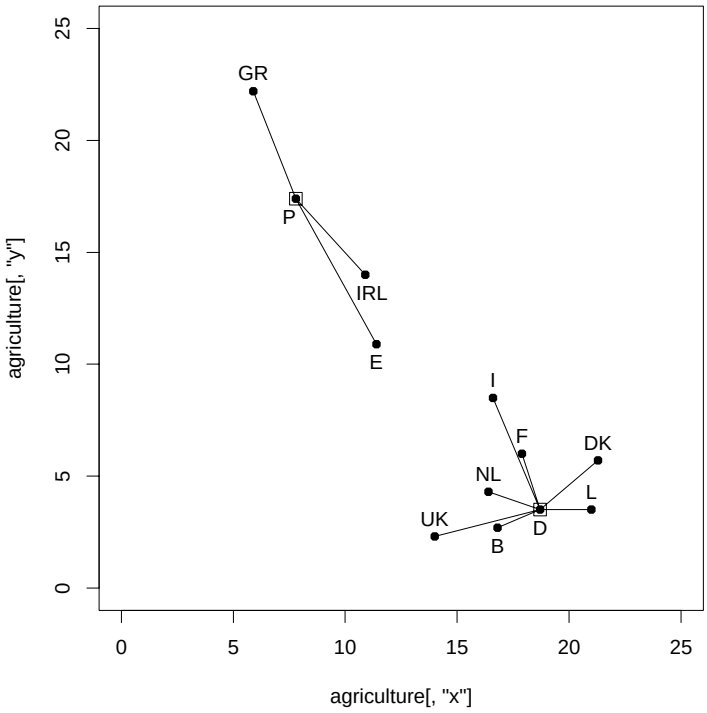


Figure 1: Agriculture dataset about 12 European countries. Applying the function `pam` with $k = 2$ yields two medoid objects, indicated by squares. Each object is then assigned to its nearest medoid, yielding clusters with 4 and 8 objects.

Each object is then assigned to the cluster corresponding to the nearest medoid. That is, object i is put into cluster v_i when medoid m_{v_i} is nearer to i than any other medoid m_w, or

$$d(i, m_{v_i}) \leq d(i, m_w) \text{ for all } w = 1, ..., k.$$

As an illustration, let us consider the agriculture dataset in Figure 1. The data were obtained from Eurostat (1994), the European statistical agency. For each country belonging to the European Union (during 1993), x_i denotes its gross national product (GNP) per capita, and y_i is the percentage of the population employed in agriculture (the data will be tabulated in Section 2.3). Applying `pam` with $k = 2$ to these data yielded the medoids Portugal and Germany, marked by squares in Figure 1. On the one hand, this determines the two clusters, since Greece, Portugal, Ireland and Spain are closer (have a shorter line) to Portugal than to Germany, whereas for the 8 remaining countries the opposite holds. On the other hand, the medoids Portugal and Germany were not chosen arbitrarily: they are such that the total length of all the lines in Figure 1 is minimal.

The actual algorithm of `pam` proceeds in two steps:

1. Step BUILD

   Construct initial 'medoids':

   - m_1 is the object with the smallest $\sum_{i=1}^{n} d(i, m_1)$
   - m_2 decreases the objective (2.1) as much as possible

     $\vdots$
   - m_k decreases the objective (2.1) as much as possible

2. Step SWAP

   Repeat until convergence:

   Consider all pairs of objects (i, j) with

   $$i \in \{m_1, \ldots, m_k\} \qquad \text{and} \qquad j \notin \{m_1, \ldots, m_k\}$$

   and make the $i \leftrightarrow j$ swap (if any) which decreases the objective most.

Since the objective function (2.1) only depends on dissimilarities between objects, the function `pam` only needs a dissimilarity matrix. When the input consists of an $n \times p$ data matrix, `pam` will first convert it to a dissimilarity matrix.

The `pam` method can be compared to the well-known *k-means* method (e.g. MacQueen 1967) which is extensively studied in the literature and is implemented in S-PLUS by the `kmeans` function using an algorithm of Hartigan and Wong (1979). In the k-means method the center of each cluster is defined as the mean (average) of all objects of the cluster. Thus `kmeans` requires the input of an $n \times p$ data matrix, unlike `pam`. The goal of `kmeans` is to minimize a sum of squared euclidean distances, implicitly assuming that each cluster has a spherical normal distribution. The function `pam` is more robust because it minimizes a sum of unsquared dissimilarities. Moreover `pam` does not need initial guesses for the cluster centers, contrary to `kmeans`. To illustrate `pam`'s robustness compared to `kmeans`, we used both methods to divide the dataset given in Figure 2 into two clusters. The result of `pam` is an almost natural division of the dataset into two clusters, indicated by a solid line on the figure, while the division proposed by `kmeans`, represented by a dashed line, is much more artificial. Also the centers found by both methods are indicated on the plot. Clearly `pam`'s medoids (the squares in Figure 2) are much closer to the real cluster centers than the means (marked by triangles in the plot) obtained by `kmeans`, although the initial guesses we gave as argument to `kmeans` were very close to the real centers.

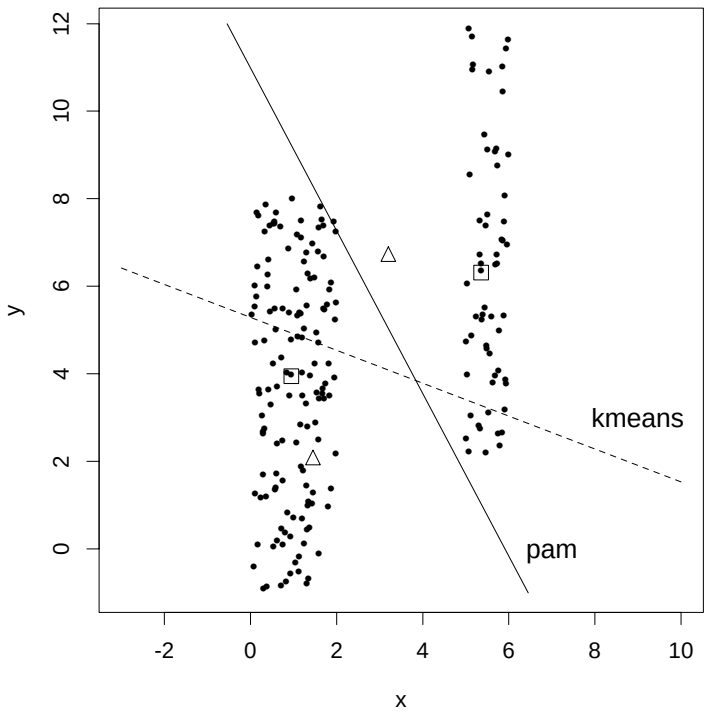


Figure 2: Division of a generated dataset into two clusters, both by `kmeans` (dashed line) and `pam` (solid line). The medoids found by `pam` are indicated by squares, and the centers found by `kmeans` are marked by triangles.

Finally `pam` provides a novel graphical display, the **silhouette plot** (Rousseeuw 1987), and a corresponding quality index allowing to select the number of clusters. Let us first explain the silhouette plot. For each object i we denote by A the cluster to which it belongs, and compute

$$\begin{aligned} a(i) &:= \frac{1}{|A|-1} \sum_{j \in A, j \neq i} d(i,j) \\ &= \text{average dissimilarity of } i \text{ to all other objects of } A. \end{aligned} \tag{2.2}$$

Now consider any cluster C different from A and put

$$\begin{aligned} d(i,C) &:= \frac{1}{|C|} \sum_{j \in C} d(i,j) \\ &= \text{average dissimilarity of } i \text{ to all objects of } C. \end{aligned} \tag{2.3}$$

After computing $d(i,C)$ for all clusters $C \neq A$ we take the smallest of those:

$$b(i) := \min_{C \neq A} d(i,C).$$

The cluster B which attains this minimum [that is, $d(i, B) = b(i)$] is called the *neighbor* of object i. This is the second-best cluster for object i.

The *silhouette value* $s(i)$ of the object i is defined as

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}.$$

Clearly $s(i)$ always lies between -1 and 1. The value $s(i)$ may be interpreted as follows:

$$\begin{aligned} s(i) \approx 1 &\Rightarrow \text{object } i \text{ is well classified (in } A); \\ s(i) \approx 0 &\Rightarrow \text{object } i \text{ lies intermediate between two clusters } (A \text{ and } B); \\ s(i) \approx -1 &\Rightarrow \text{object } i \text{ is badly classified (closer to } B \text{ than to } A). \end{aligned}$$

The silhouette of the cluster A is a plot of all its $s(i)$, ranked in decreasing order. The entire silhouette plot shows the silhouettes of all clusters below each other, so the quality of the clusters can be compared: a wide (dark) silhouette is better than a narrow one.

For instance, Figure 3 shows the silhouette plot of the clustering in Figure 1. It turns out that the medoids (P and D) have a high $s(i)$ in their cluster, whereas the boundary object E (Spain) has a low $s(i)$.

The quality index mentioned earlier is the *overall average silhouette width* of the silhouette plot, defined as the average of the $s(i)$ over all objects i in the dataset. In Figure 3 we find the value 0.63, indicating that a reasonable structure has been found.

In general we propose to run `pam` several times, each time with a different k, and to compare the resulting silhouette plots. The user can then select that value of k yielding the highest average silhouette width, over all k, which we call the *silhouette coefficient*. Experience has led to the subjective interpretation of the silhouette coefficient (SC) as listed in Table 2. This interpretation does not depend on the number of objects.

SC	Proposed Interpretation
0.71–1.00	A strong structure has been found.
0.51–0.70	A reasonable structure has been found.
0.26–0.50	The structure is weak and could be artificial, try additional methods.
≤ 0.25	No substantial structure has been found.

Table 2: Interpretation of the silhouette coefficient for partitioning methods.

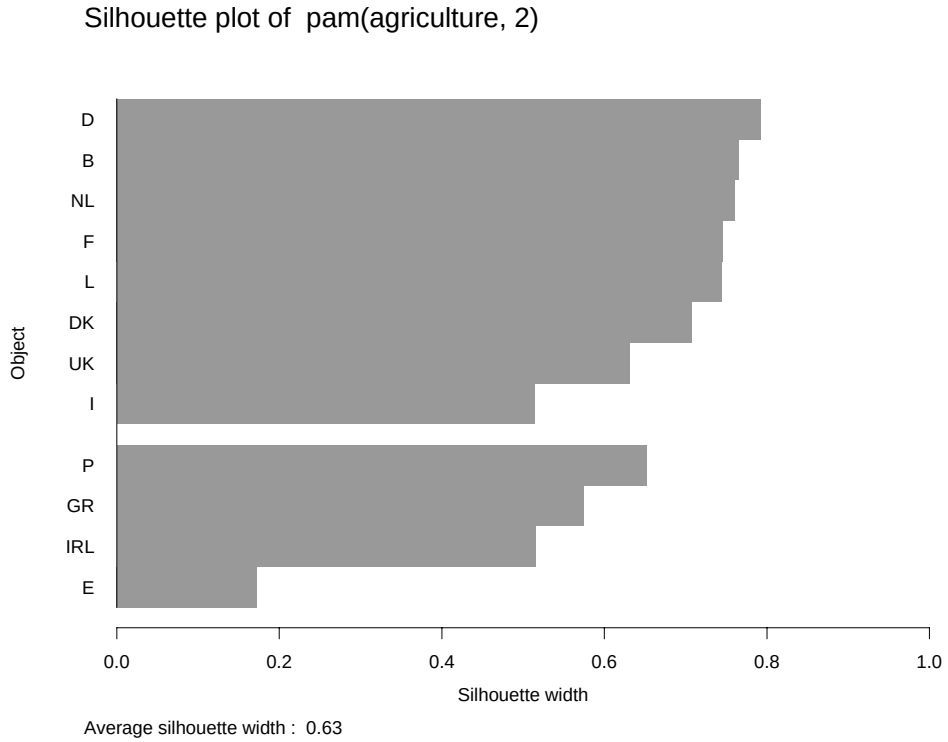


Figure 3: Silhouette plot of the `pam` clustering of Figure 1.

In S-PLUS, the call to `pam` is

```
pam(x, k, diss = F, metric = "euclidean", stand = F).
```

The function's arguments have the following meaning:

- `x` and `diss`: the argument `x` contains the input data. `diss` tells `pam` whether `x` should be interpreted as a data matrix (`diss=F`), or as a vector representing a dissimilarity matrix (`diss=T`), the default being a data matrix of objects-by-variables. In this case, all variables must be numeric, otherwise the function `daisy` should be used to compute the dissimilarities. When a dissimilarity matrix is given as input (`diss=T`), it is preferably an object of class `dissimilarity`. However, `pam` will also accept dissimilarities produced with the existing S-PLUS function `dist`, or any vector that can be interpreted as a dissimilarity matrix.

- `k`: the number of clusters to look for.

- `metric` and `stand`: these arguments are only used when `diss=F`, and then they behave in the same way as the corresponding arguments of `daisy` (see Section 1).

The output of the function `pam` is an object of class `pam`. It contains the same numerical

information that results from the Fortran program, such as the medoids, the size of the clusters, their silhouette width etc. All details about the input arguments and the structure of the output can be found in our S-PLUS help files. Additional examples and information are given in [KR, chapter 2].

The `pam` class inherits from the `partition` class, which can be given as input to the high level graphics function `plot`. A silhouette plot of an object of class `pam` is then made on the current graphics device. An overview of all new classes and their generic functions can be found in Figure 11. A summary of the object-oriented terminology in S-PLUS is given in the Appendix.

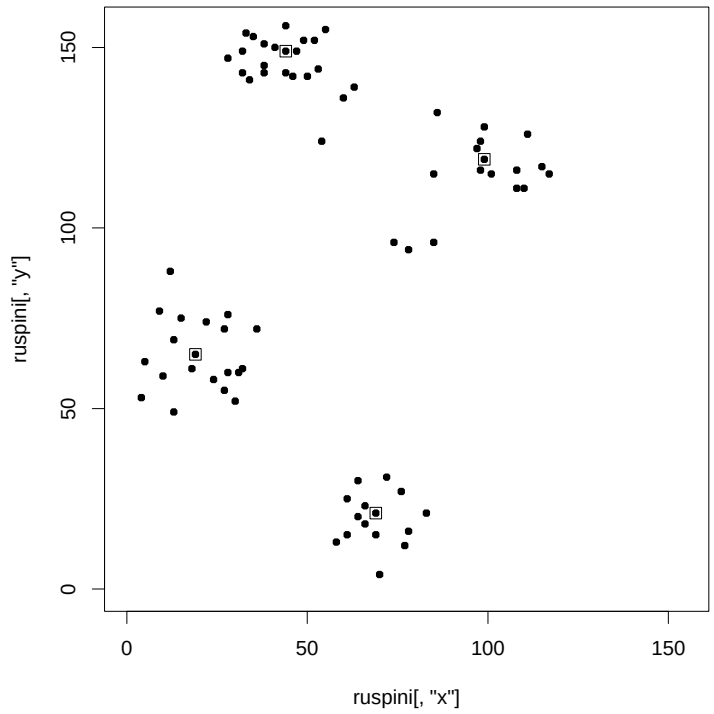
Example. Let us consider the dataset of Ruspini (1970) listed in [KR, page 100] which consists of 75 bivariate points, plotted in Figure 4a. We use `pam` to partition the data into $k = 4$ clusters. This is obtained by the S-PLUS instruction `pam(ruspini,4)`. In this way, we use the default values for `diss`, `metric`, and `stand`. The four resulting medoids are indicated by squares in Figure 4a, to illustrate that they are centrally located in the four natural clusters. It can be verified that `pam` correctly identifies these four clusters. The resulting silhouette plot is shown in Figure 4b. The silhouette widths $s(i)$ of all objects are visibly above 0.4. The average silhouette width of 0.74 shows that a good clustering structure has been found. We have also partitioned the Ruspini data by `pam` into 3 and 5 clusters. The resulting average silhouette widths were 0.63 and 0.71 respectively. Therefore, we prefer the partition into 4 clusters.

2.2 Clustering Large Applications: function `clara`

The function `pam` needs to store the entire dissimilarity matrix with its $O(n^2)$ entries in central memory, hence its computation time goes up at least as fast. For larger datasets (say, with more than 250 objects) this becomes impractical. To avoid this problem, Kaufman and Rousseeuw (1986) introduced the method `clara` which does not store the entire dissimilarity matrix. Therefore, `clara` only accepts input of an $n \times p$ data matrix. The `clara` algorithm proceeds as follows, with storage and time complexity indicated at the right:

- Input of n cases with measurements. $O(n)$
- Repeat '`samples`' times:
  - draw a subdataset of '`sampsize`' cases fixed

(a)



(b)

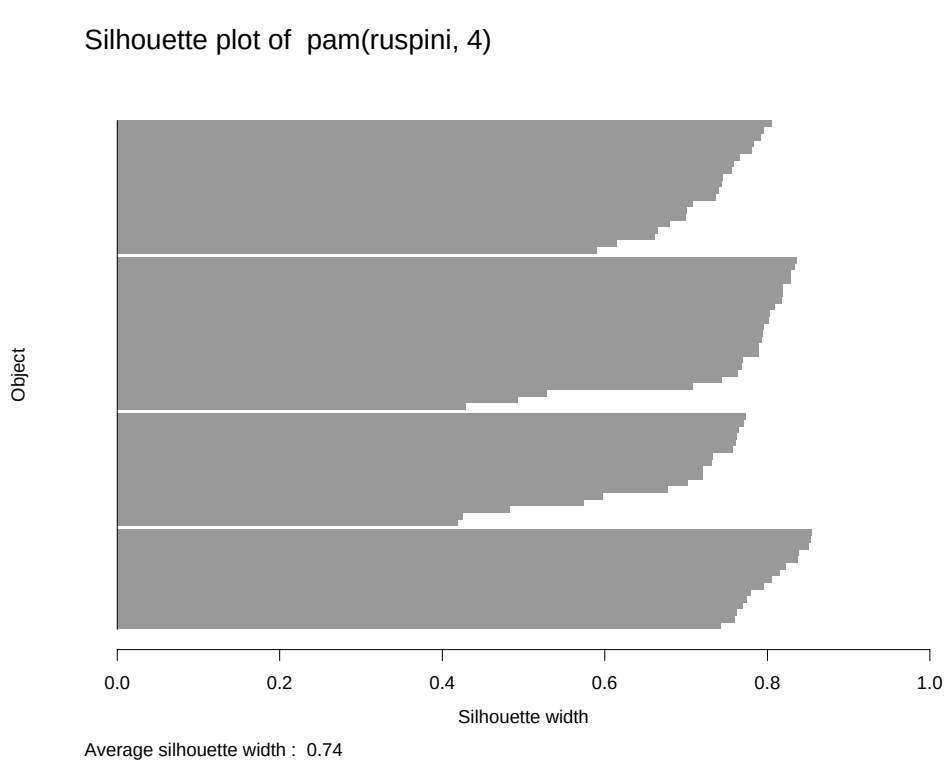


Figure 4: (a) Plot of the Ruspini data. The observations indicated by squares are the medoids found by `pam` for $k = 4$; (b) silhouette plot of the partition obtained by `pam`.

  - applying `pam` to it yields medoids $\{m_1, \ldots, m_k\}$ fixed
  - compute objective $= \sum_{i=1}^{n} \min_{t=1,\ldots,k} d(i, m_t)$ O(n)
  - retain $\{m_1, \ldots, m_k\}$ if objective $<$ currently best objective. fixed
- Assigning all n cases to $\{m_1, \ldots, m_k\}$ yields the final partition. O(n)

The total storage and time are thus linear in n, instead of quadratic!

The clustering obtained by `clara` can also be represented by means of a silhouette plot, as described in Section 2.1. Due to the size of the dataset, the silhouette plot is given only for the best subdataset.

In S-PLUS, the header of the function `clara` is

```
clara(x, k, metric = "euclidean", stand = F,
      samples = 5, sampsize = 40+2*k).
```

The available arguments are:

- `x`: the input data. Note that the argument `diss` is not available here since storing a huge dissimilarity matrix is impossible. `x` has to be a matrix or dataframe of objects-by-variables.
- `k`: the number of clusters to look for.
- `metric` and `stand`: like the corresponding arguments of `daisy` (Section 1).
- `samples`: the number of subdatasets that are drawn from the original dataset.
- `sampsize`: the number of objects in each subdataset.

The last two arguments were fixed numbers in the Fortran program, but have now been made variable. The output of this function is an object of class `clara`. Whereas the original Fortran program gave intermediate output for each subdataset, the output is now restricted to the final clustering. The `clara` class inherits from the `partition` class, hence the generic function `plot` yields the silhouette plot.

Example. A dataset with 3 clusters of 1000 bivariate objects each was generated in S-PLUS. We called `clara(xclara,3)` to cluster the data. We did not specify the number of subdatasets or their size, thus accepting the default values as defined in the function header (5 samples of size $40+2*k=46$). Figure 5a plots the data, with three lines that separate the clusters found by `clara`. (A point in such a sector is closest to the corresponding medoid.) The silhouette plot in Figure 5b is quite pronounced and possesses a high overall average silhouette width of 0.74. This indicates that `clara` has found a strong clustering structure.

2.3 Fuzzy Analysis: function `fanny`

The functions `pam` and `clara` are *crisp* clustering methods. This means that each object of the dataset is assigned to exactly one cluster. For instance, an object lying between two clusters will be assigned to one of them. However, a *fuzzy* method spreads each object over the various clusters. For each object i and each cluster v there will be a *membership* u_{iv} which indicates how strongly object i belongs to cluster v. Memberships have to satisfy the following conditions:

- $u_{iv} \ \geq \ 0$ for all $i = 1, \ldots, n$ and all $v = 1, \ldots, k$.
- $\sum_{v=1}^{k} \ u_{iv} \ = \ 1 \ = \ 100\%$ for all $i = 1, \ldots, n$.

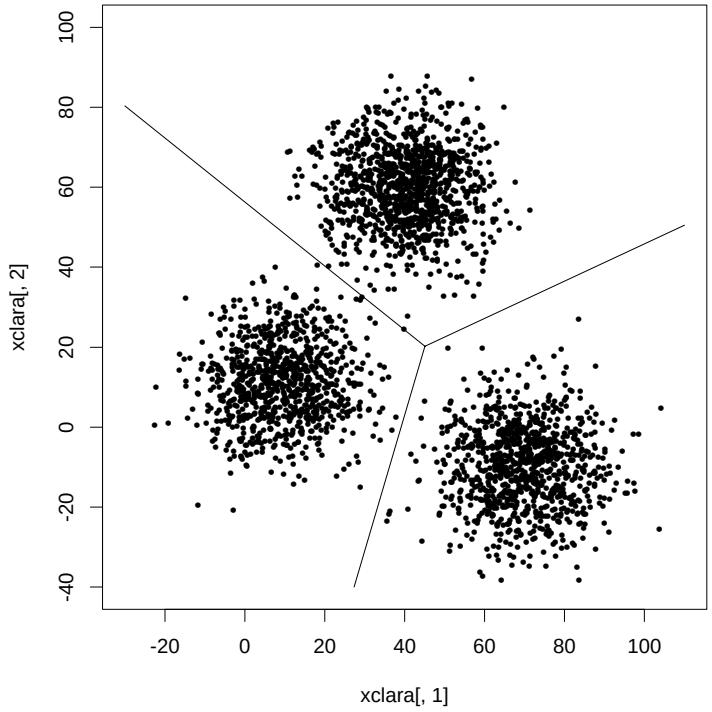
To illustrate memberships, let us consider Figure 1. The crisp clustering method `pam` has assigned each object to exactly one cluster. In other words, it has produced memberships that are either zero or one, as shown in Table 3. On the other hand, a fuzzy clustering method yields memberships that can take on any value between zero and one, as in the rightmost column of Table 3 where we see that the "intermediate" country Spain (E) belongs for 31% to cluster 1 and for 69% to cluster 2. The other fuzzy memberships are more clear-cut.

Here we will focus on the method `fanny` proposed in [KR, chapter 4] where the memberships u_{iv} are defined through minimization of the objective function

$$\sum_{v=1}^{k} \ \frac{\sum_{i,j=1}^{n} \ u_{iv}^2 u_{jv}^2 d(i,j)}{2 \sum_{j=1}^{n} \ u_{jv}^2}. \tag{2.4}$$

In this expression, the dissimilarities $d(i,j)$ are known and the memberships u_{iv} are unknown. The minimization is carried out numerically by means of an iterative algorithm, taking into account the side constraints on memberships by means of Lagrange multipliers.

(a)



(b)

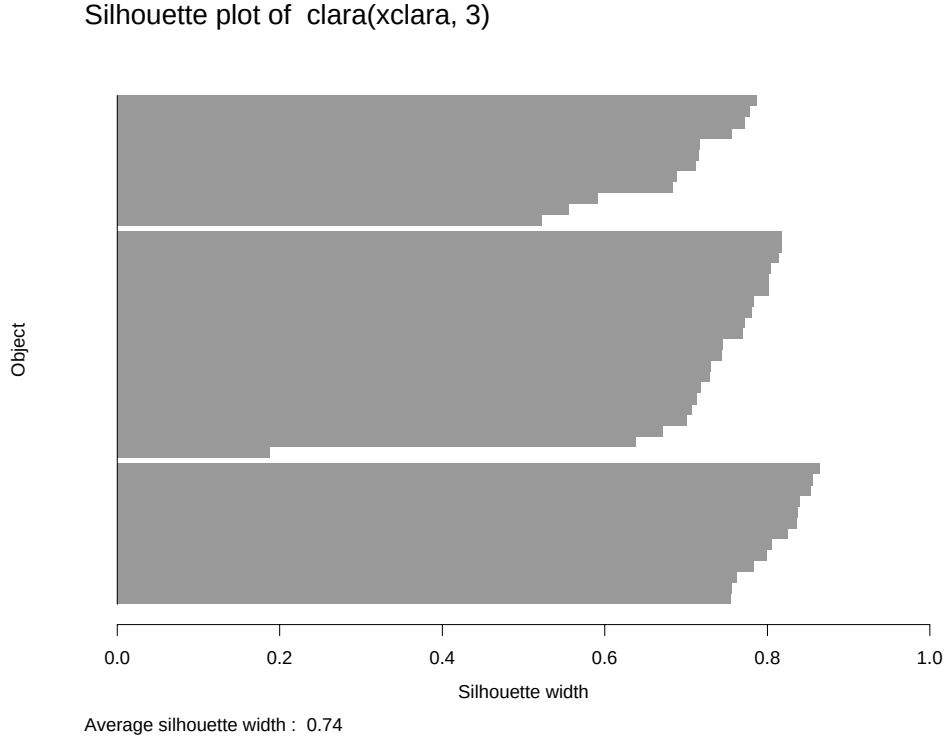


Figure 5: (a) plot of 3000 observations and its partition into $k = 3$ clusters given by `clara`; (b) corresponding silhouette plot.

			crisp memberships		fuzzy memberships	
Country	x_i	y_i	u_{i1}	u_{i2}	u_{i1}	u_{i2}
B (Belgium)	16.8	2.7	1	0	.90	.10
DK (Denmark)	21.3	5.7	1	0	.85	.15
D (Germany)	18.7	3.5	1	0	.93	.07
GR (Greece)	5.9	22.2	0	1	.18	.82
E (Spain)	11.4	10.9	0	1	.31	.69
F (France)	17.8	6.0	1	0	.90	.10
IRL (Ireland)	10.9	14.0	0	1	.15	.85
I (Italy)	16.6	8.5	1	0	.73	.27
L (Luxemburg)	21.0	3.5	1	0	.87	.13
NL (Netherlands)	16.4	4.3	1	0	.91	.09
P (Portugal)	7.8	17.4	0	1	.10	.90
UK (U.Kingdom)	14.0	2.3	1	0	.79	.21

Table 3: Agriculture dataset about 12 European countries, with crisp memberships (obtained by `pam`) and fuzzy memberships (obtained by `fanny`).

Compared to other fuzzy clustering methods, `fanny` has the advantage that it can handle dissimilarity data, since (2.4) only uses inter-object dissimilarities and does not involve any averages of objects (see Rousseeuw 1995). Also, `fanny` is rather robust to the assumption of spherical clusters since the $d(i, j)$ in (2.4) are not squared.

For any fuzzy clustering, such as the one produced by `fanny`, one can consider the *nearest crisp clustering*. The latter assigns each object i to the cluster v in which it has the highest membership u_{iv}. This crisp clustering can then be represented by a silhouette plot. In the example of Table 3, the crisp clustering closest to that of `fanny` happens to be that of `pam`, hence yielding the same silhouette plot.

In S-PLUS the function `fanny` is invoked like `pam`, namely

```
fanny(x, k, diss = F, metric = "euclidean", stand = F).
```

The output of the function `fanny` is an object of class `fanny`. It contains e.g. the memberships of all objects and the nearest crisp clustering. The class `fanny` inherits from the `partition` class, therefore the generic function `plot` yields the silhouette plot.

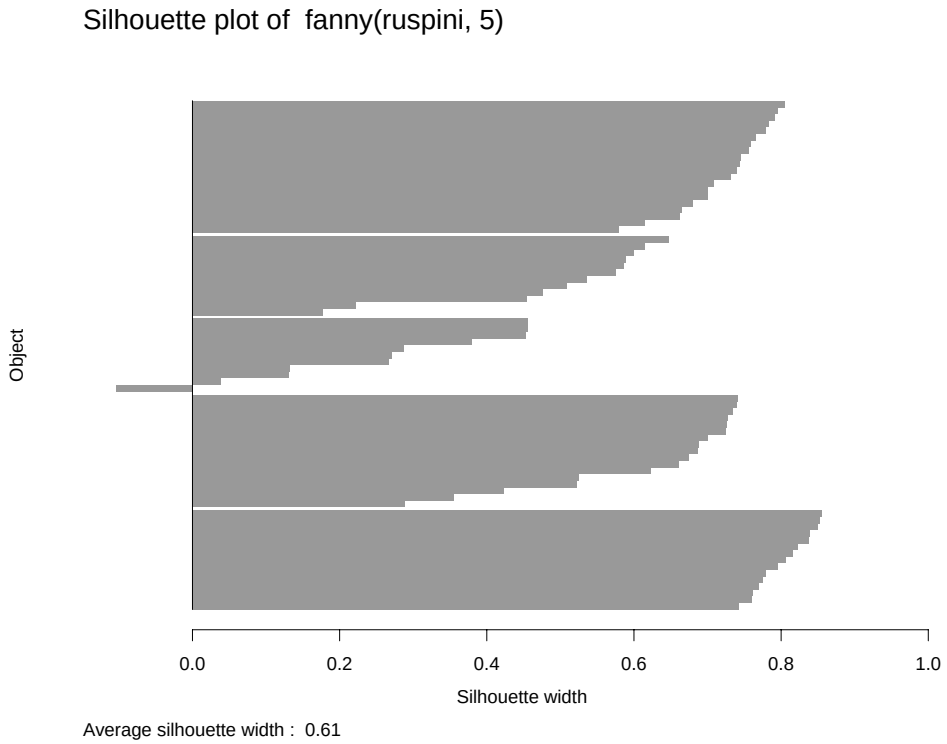


Figure 6: Silhouette plot of the partition obtained by `fanny` with $k = 5$.

Example. We first use `fanny` to partition the Ruspini data into $k = 4$ clusters. The nearest crisp clustering then equals that of `pam`, hence the silhouette plots are identical. But when we call `fanny` with $k = 5$, using the S-PLUS command `fanny(ruspini,5)`, the nearest crisp clustering differs from that of `pam`. The silhouette plot (Figure 6) shows that `fanny` splits the second cluster into two smaller clusters, and that some "bridging" objects in the new clusters have very small or even negative $s(i)$, meaning that their memberships to both new clusters are approximately the same. Consequently, the average silhouette width (0.61 for $k = 5$ compared to 0.74 for $k = 4$) is punished for the "artificial" splitting of a cluster.

3 Hierarchical methods

3.1 Agglomerative Nesting: function `agnes`

The function `agnes` is of the agglomerative hierarchical type, hence it yields a sequence of clusterings. In the first clustering each of the n objects forms its own separate cluster. In subsequent steps clusters are merged, until (after $n - 1$ steps) only one large cluster

remains. Many such methods exist. In `agnes`, the group average method is taken as the default, based on arguments of robustness, monotonicity and consistency [KR, pages 238–243]. Also four other well-known methods are available in `agnes`, namely single linkage, complete linkage, Ward's method, and weighted average linkage. These five methods can be described in a unified way (Lance and Williams 1966) and are already implemented in many software packages. We did not add Gower's method nor the centroid method because their dissimilarities between merging clusters are not necessarily monotone, which makes the resulting graphical displays very confusing.

Compared to other implementations such as the function `hclust` in S-PLUS, `agnes` has two additional features: (a) it yields the *agglomerative coefficient* which measures the amount of clustering structure found; and (b) apart from the usual tree it also provides the *banner*, a novel graphical display.

The agglomerative coefficient (Rousseeuw 1986) is a quality index for an agglomerative clustering of the data. For each object i denote by $d(i)$ its dissimilarity to the first cluster it is merged with, divided by the dissimilarity of the merger in the last step of the algorithm. The agglomerative coefficient (AC) is then defined as the average of all $1 - d(i)$. Note that the AC tends to increase with the number of objects, unlike the average silhouette width of Section 2.

The hierarchy obtained from `agnes` can be graphically displayed in two ways: by means of a clustering tree or by a banner.

Agglomerative tree: A tree in which the leaves represent objects. The vertical coordinate of the junction of two branches is the dissimilarity between the corresponding clusters. An agglomerative clustering tree is a rotated version of a dendrogram, as it was programmed by Anderberg (1973) and many others.

Agglomerative banner: The banner shows the successive mergers from left to right. (Imagine the ragged flag parts at the left, and the flagstaff at the right.) The objects are listed vertically. The merger of two clusters is represented by a horizontal bar which commences at the between-cluster dissimilarity. The banner thus contains the same information as the clustering tree. Note that the agglomerative coefficient (AC) defined above can be seen as the average width (the percentage filled) of the banner.

In S-PLUS, the call to `agnes` is of the form

`agnes(x, diss = F, metric = "euclidean", stand = F, method = "average").`

The arguments are similar as for `pam` and `fanny`, of course without the number of clusters. The `method` argument determines which method is to be used:

- `average`: group average method.
- `single`: single linkage.
- `complete`: complete linkage.
- `ward`: Ward's method.
- `weighted`: weighted average linkage.

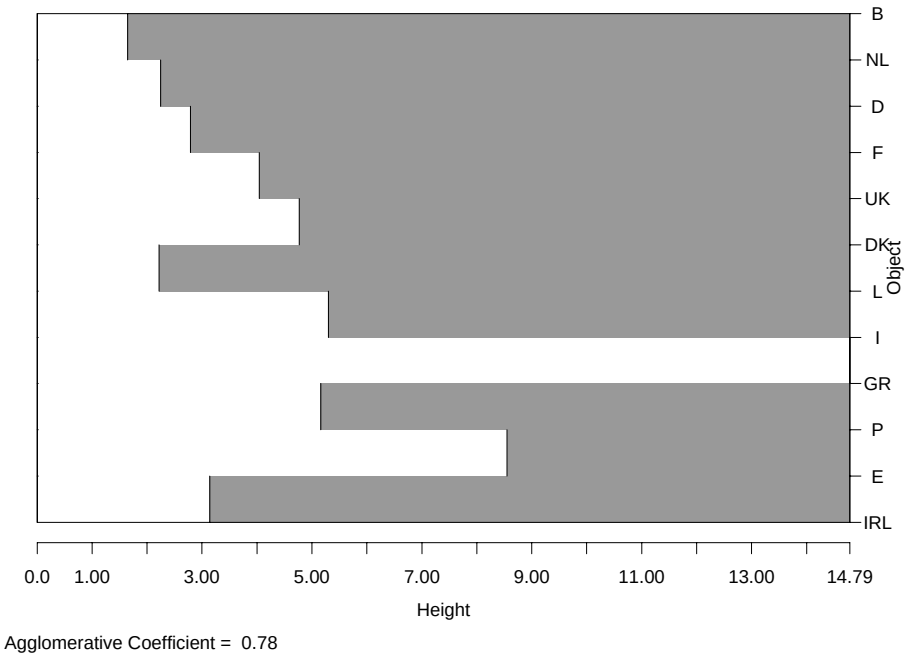
The output of `agnes` is an object of class `agnes`, which contains e.g. an ordering of the objects in which they can be plotted, the distances between clusters at the successive mergers, and the agglomerative coefficient. An `agnes` object can be given as input to the high level graphics functions `plot` and `pltree`. The new `plot.agnes` method draws a banner of an object of class `agnes` on the current graphics device, whereas `pltree.agnes` produces the clustering tree.

Example. We invoke `agnes` with the call `agnes(agriculture)`, thus accepting all default settings. In the agriculture dataset `agnes` ends up with a cluster of 8 countries that are less oriented towards agriculture and whose gross national product is relatively high, and a cluster of 4 countries in the opposite situation. The agglomerative coefficient is high (0.78) which indicates a good clustering structure. Both the banner (Figure 7a) and the clustering tree (Figure 7b) have been plotted, to facilitate comparisons between them.

3.2 Divisive Analysis: function `diana`

The function `diana` is a divisive hierarchical method. The initial clustering (at step 0) consists of one large cluster containing all n objects. In each subsequent step, the largest available cluster is split into two smaller clusters, until finally all clusters contain but a single object. The function `diana` is probably unique in computing a divisive hierarchy, whereas most other software for hierarchical clustering is agglomerative.

(a)



(b)

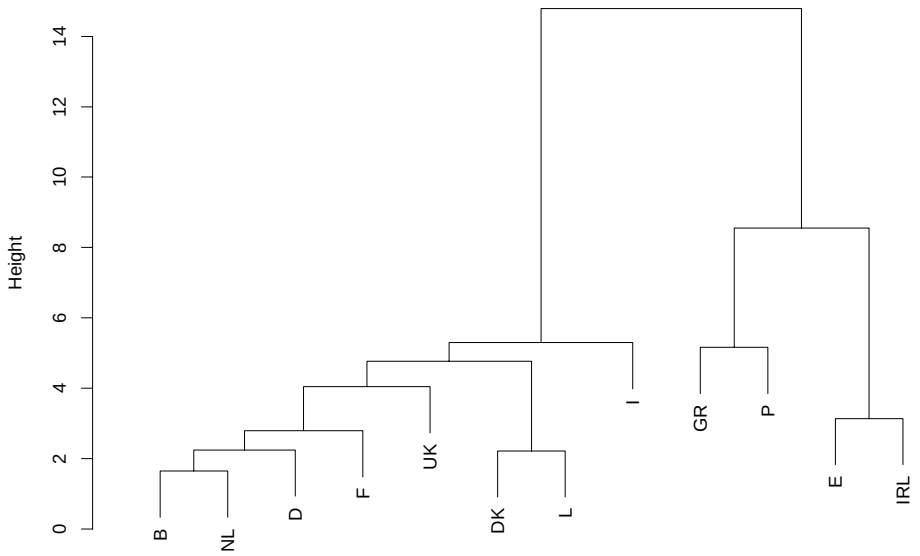


Figure 7: Result of applying `agnes` to the agriculture data, represented in the form of (a) a banner; and (b) a clustering tree.

The complete algorithm consists of $n-1$ successive splits. In each step, we select the cluster C with the largest diameter, where

$$\text{diam}(C) := \max_{i,j \in C} d(i,j).$$

Assuming $\text{diam}(C) > 0$ we then split up C into two clusters A and B, according to a variant of the method of Macnaughton-Smith et al. (1964). Below we describe in pseudocode how such a split is performed. At first $A := C$ and $B := \varnothing$, and then

1. Move one object from A to B.
   For each object $i \in A$ we calculate $a(i)$, the average dissimilarity to all other objects of A, as in (2.2). The object m of A for which $a(m)$ is the largest, is moved to B:
   $$A := A \setminus \{m\},\ B := \{m\}.$$

2. Move other objects from A to B, which is called the "splinter group".
   If $|A| = 1$, stop. Otherwise calculate $a(i)$ for all $i \in A$, and the average dissimilarity of i to all objects of B, denoted as $d(i,B)$ in (2.3). Select the object $h \in A$ for which $a(h) - d(h,B) = \max_{i \in A}(a(i) - d(i,B))$.
   If $a(h) - d(h,B) > 0 \Rightarrow$ move h from A to B, and go to 2.
   If $a(h) - d(h,B) \leq 0 \Rightarrow$ the process stops. Keep A and B as they are now.

The function `diana` also provides the *divisive coefficient* (Rousseeuw 1986), which measures the clustering structure of the dataset. This coefficient is obtained as follows: for each object i, denote by $d(i)$ the diameter of the last cluster to which it belongs (before being split off as a single object), divided by the diameter of the whole dataset. The divisive coefficient (DC) is then defined as the average of all $d(i)$. The DC tends to be slightly larger than the AC of `agnes`. Like the AC, also the DC grows with the number of objects. Therefore, neither the AC or the DC can be used to compare datasets of very different sizes.

As with `agnes`, the hierarchy obtained from `diana` can be displayed by means of a tree or a banner.

Divisive tree: Here the stem represents the entire dataset. The vertical coordinate where a branch splits in two equals the diameter of that cluster before splitting.

Divisive banner: The banner shows the successive splits from left to right. (Imagine the flagstaff at the left, and the ragged endings at the right.) The objects are stitched

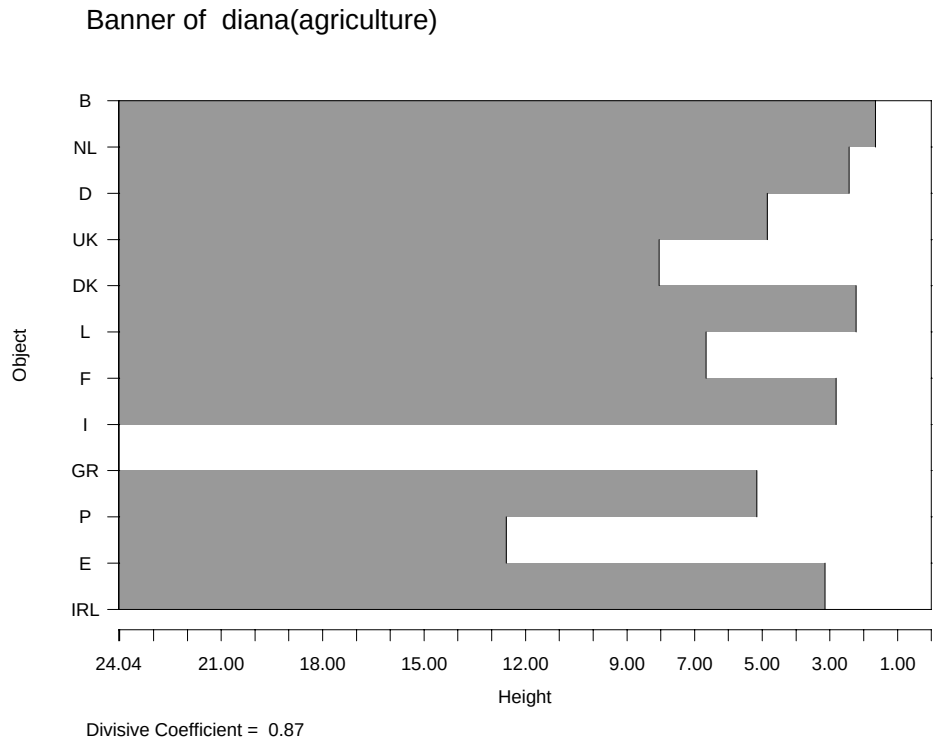


Figure 8: Result of applying `diana` to the agriculture data, represented in a banner.

together by horizontal bars, which end at the diameter of the cluster being split. The divisive coefficient (DC) defined above equals the average width of the banner.

In S-PLUS, `diana` is called by

```
diana(x, diss = F, metric = "euclidean", stand = F).
```

The arguments are as in `agnes`, but we no longer have "`method`". The output of `diana` is an object of class `diana`, which contains e.g. an ordering of the objects to allow for plotting, the diameters of the clusters prior to splitting, and the divisive coefficient. A `diana` object can be inputted to the high level graphics functions `plot` and `pltree`. The method `plot.diana` produces the divisive banner, whereas `pltree.diana` yields the clustering tree.

Example. To make the result comparable to that of `agnes`, we again use the default settings (no standardization and Euclidean metric), and we call `diana` as `diana(agriculture)`. In the agriculture dataset `diana` finds essentially the same clustering structure as `agnes`, with a high divisive coefficient (0.87). The divisive banner in Figure 8 is almost a mirror image of the agglomerative banner in Figure 7a.

3.3 Monothetic Analysis: function `mona`

The function `mona` is a different type of divisive hierarchical method. Contrary to `diana`, which can process a dissimilarity matrix as well as a data matrix with numeric variables, `mona` operates on a data matrix with binary variables. For each split `mona` uses a single (well-chosen) variable, which is why it is called a *monothetic* method. Most other hierarchical methods (including `agnes` and `diana`) are *polythetic*, i.e. they use all variables simultaneously.

Consider the example in Figure 9. At step 0 we have one cluster consisting of all objects. In the first step `mona` selects one particular binary variable (v_4), and divides the data in a cluster where $v_4 = 1$ and another cluster where $v_4 = 0$. At each subsequent step, `mona` splits all available clusters. The variable used for splitting a cluster is selected by a robust version of the method of Williams and Lambert (1959); for further information see [KR, chapter 7]. Note that in Figure 9 the cluster $\{C, D\}$ cannot be split anymore, which means that C and D have the same value for each binary variable.

The clustering hierarchy constructed by `mona` can also be represented by means of a divisive banner in Figure 9b. The length of a bar is now given by the number of divisive steps needed to make that split. Inside the bar, the variable is listed which was responsible for the split. A bar continuing to the right margin (here, to the imaginary "step 4") indicates a cluster that cannot be split.

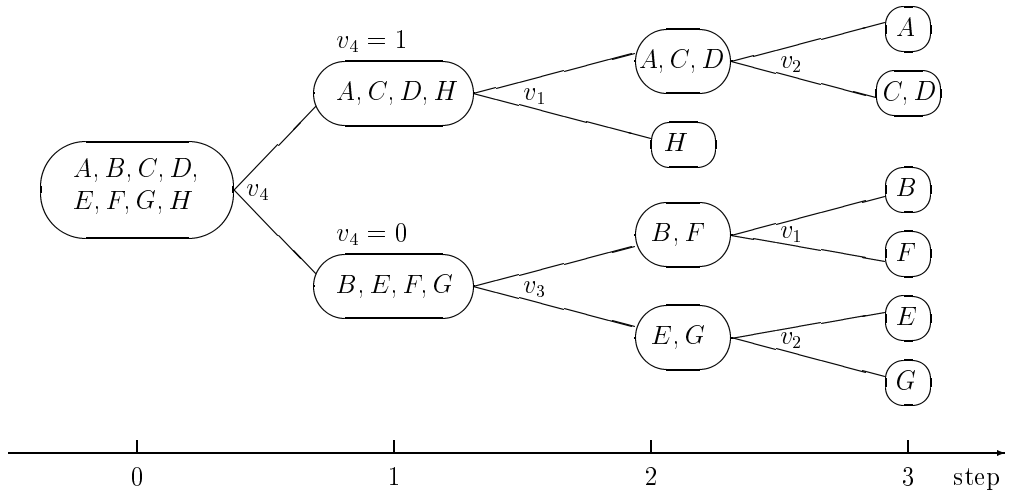
In S-PLUS, the call to `mona` is simply

```
mona(x).
```

The input argument `x` should be a matrix or dataframe containing only binary variables. These can be factors with two levels, as well as numeric or character variables which only take on two values. The output of this function is an object of class `mona` which among other things contains the revised dataset, with any missing values replaced by estimated values. Information about how this was done can be found in our accompanying S-PLUS help files and [KR, pages 298–299]. A `mona` object can be inputted to the high level graphics function `plot`, in which case `plot.mona` draws the banner.

Example. The animals dataset is described in [KR, page 295]. Six binary attributes are considered for twenty animals. The banner in Figure 10 shows how `mona` classifies the animals according to the attributes. In the first step, cold- and warm-blooded animals are

(a)



(b)

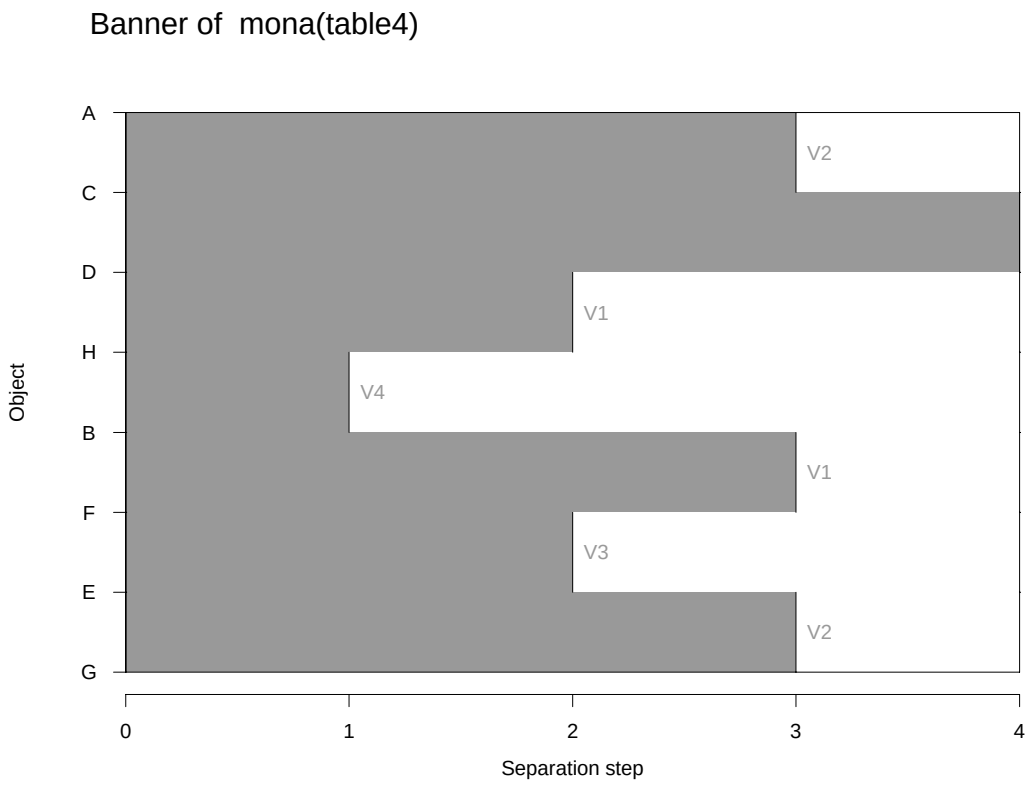


Figure 9: Clustering hierarchy obtained by `mona` for 8 objects with 4 binary variables: (a) depicted as a tree; (b) in the form of a banner.

put in separate clusters. The first cluster is then split into vertebrates and invertebrates, and the second cluster into flying and non-flying animals. The other three variables are gro (live in groups), hai (have hair) and end (are endangered). After the fifth step, animals belonging to the same group have the same value for all variables, so in the banner long bars are drawn that keep these clusters intact.

4 Conclusions

We have developed several new S-PLUS functions to perform cluster analysis. Special attention was paid to the uniformity of their input and output structure, and their compatibility with existing S-PLUS functions. Figure 11 shows the interplay of the new functions in this object-oriented environment. The various classes are displayed inside ovals, and they are connected with the available generic functions. Note that the generic function `summary` provides much more extensive output than `print`.

The new algorithms have several statistical advantages over some customary clustering methods. First, they are more robust with respect to outliers and other violations of the assumption of spherically normal clusters, since they are all based on sums of unsquared distances (they are L^1 methods, contrary to the usual L^2 methods). Secondly, `pam`, `fanny`, `agnes` and `diana` can cluster dissimilarity data as well as objects-by-variables data, whereas most existing algorithms only accept data of the latter type. The new functions are also able to handle a certain number of missing measurements.

Finally, the new algorithms come with novel graphical displays. For the partition algorithms, the generic function `plot` yields the silhouettes. This display is especially useful for clustering dissimilarity data and datasets with more than three variables, because then the pattern is hard to see by eye. The silhouette plot shows which objects are well classified and which are intermediate. Moreover, it visualizes the quality of each cluster (how well it is separated from other clusters, compared to its compactness). Finally, the overall quality of the partition is measured by the average silhouette width. This allows the user to compare the strength of different partitions, and to select an appropriate number of clusters. For hierarchical algorithms, `pltree` provides the usual tree and `plot` yields the banner. The average width of the banner reflects the strength of the clustering, which equivalently is measured by the agglomerative coefficient or the divisive coefficient.

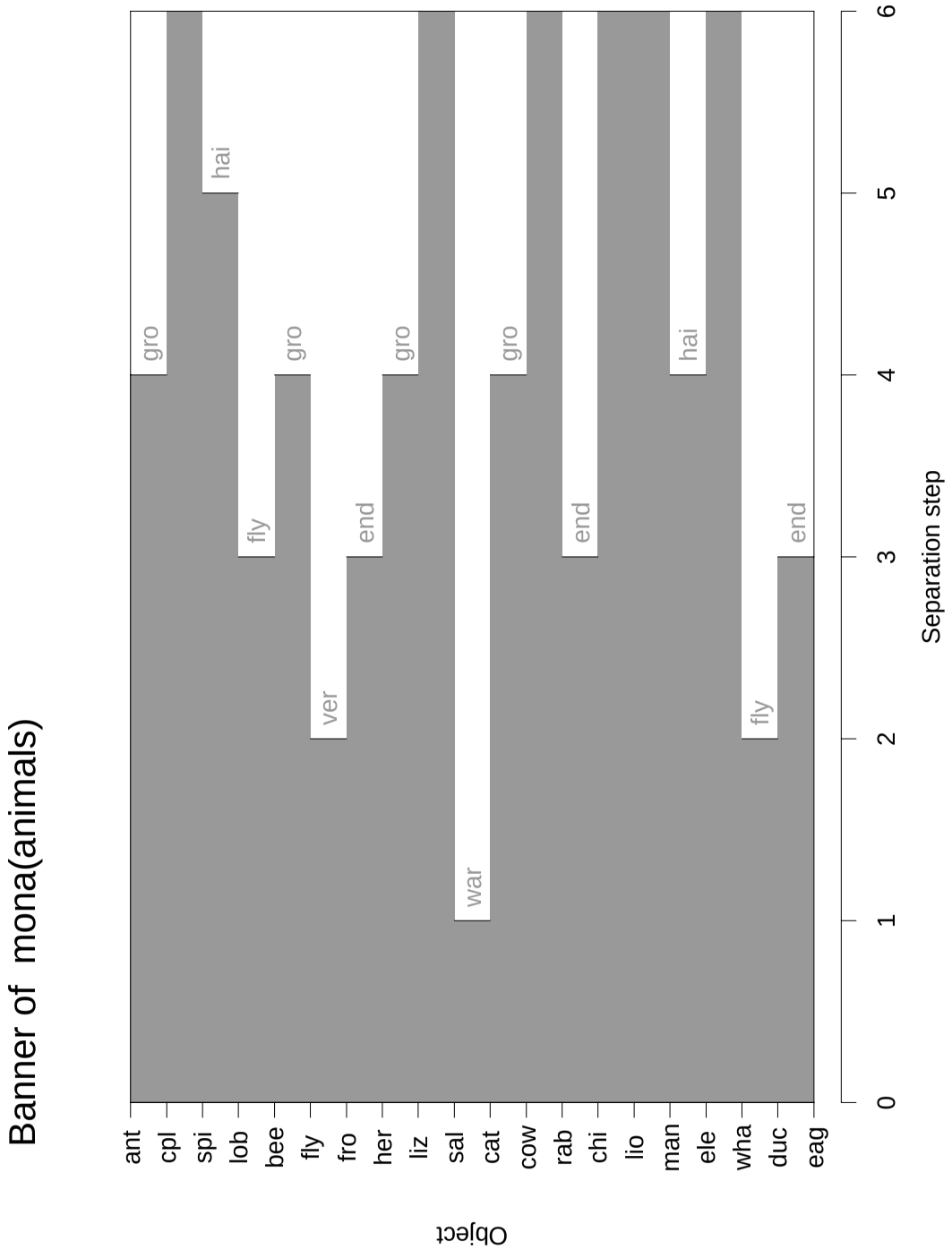


Figure 10: Animals data: divisive banner of the `mona` clustering. The clusters at the imaginary last step (here, "step 6") are unsplittable.

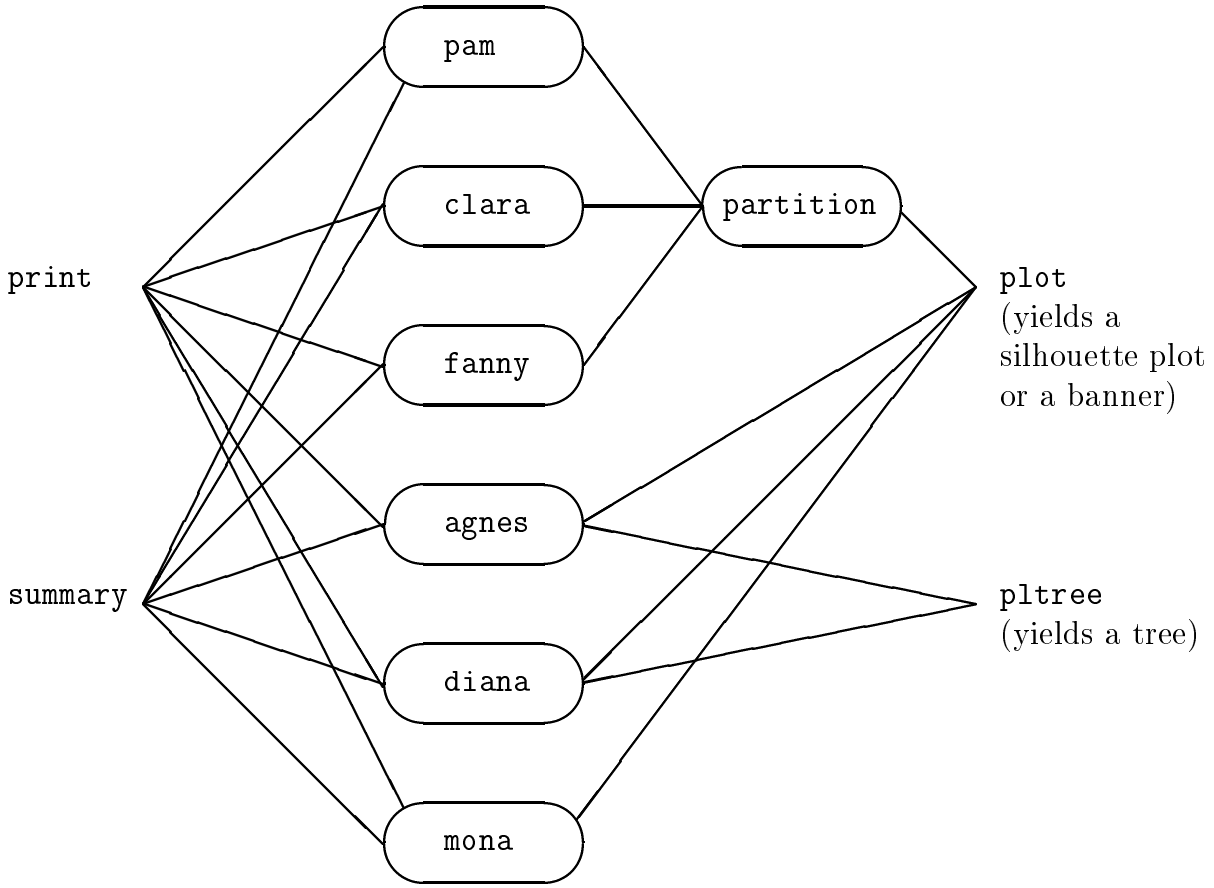


Figure 11: Overview of the implementation in S-PLUS: classes (in ovals) and four generic functions. For each line connecting a generic function and a class we wrote a new method, e.g. `print.clara`, `summary.diana`, `plot.partition`, `plot.mona`, `pltree.agnes`, etc.

5 Appendix

Object-oriented terminology in S-PLUS (adapted from Chambers and Hastie 1991, appendix A). In the object-oriented programming paradigm, the execution of a program is a set of actions that take place as the result of passing *messages* between *objects*. The messages can be thought of as requests (for example, that the object prints itself). Each object belongs to a *class*, which determines for which messages the object has a *method*. A method implements the definition of what should happen in response to the message.

In S-PLUS one does not send messages directly to an object, but one calls a *generic function* with the object as argument, for example `print(pamobject)`. This generic function finds the class of the object, and finds a special method of the generic function for that class (if one exists). Then this method is evaluated instead of the generic function itself, for

example `print.pam(pamobject)` is evaluated.

Objects belonging to a class can *inherit* methods from another class. This simplifies defining new classes that are adapted from existing classes, since only methods related to new features need to be written. For example, an object of class `fanny` inherits a method for the generic function `plot` from the class `partition`.

Questions or remarks about the implementation can be directed to
`Anja.Struyf@uia.ua.ac.be` and `Mia.Hubert@uia.ua.ac.be`.

References

Anderberg, M.R., *Cluster Analysis for Applications* (Academic, New York, 1973).

Chambers, J.M. and T.J. Hastie, *Statistical models in S* (Wadsworth, California, 1991).

Eurostat, *Cijfers en feiten: Een statistisch portret van de Europese Unie* (1994).

Gower, J.C., A general coefficient of similarity and some of its properties, *Biometrics*, **27** (1971) 857–871.

Hartigan, J.A. and M.A. Wong, A k-means clustering algorithm, *Applied Statistics*, **28** (1979) 100–108.

Kaufman, L. and P.J. Rousseeuw, Clustering large data sets (with discussion), in: E.S. Gelsema and L.N. Kanal (Eds.), *Pattern Recognition in Practice II* (North-Holland, Amsterdam, 1986) 425–437.

Kaufman, L. and P.J. Rousseeuw, Clustering by means of medoids, in: Y. Dodge (Ed.), *Statistical Data Analysis based on the L_1 Norm* (North-Holland, Amsterdam, 1987) 405–416.

[KR] Kaufman, L. and P.J. Rousseeuw, *Finding Groups in Data* (John Wiley & Sons, New York, 1990).

Lance, G.N. and W.T. Williams, A general theory of classificatory sorting strategies: 1. Hierarchical systems, *Computer J.*, **11** (1966) 195.

Macnaughton-Smith, P., W.T. Williams, M.B. Dale, and L.G. Mockett, Dissimilarity analysis: A new technique of hierarchical sub-division, *Nature*, **202** (1964) 1034–1035.

MacQueen, J., Some methods for classification and analysis of multivariate observations, in: L. Le Cam and J. Neyman (Eds.), *5th Berkeley Symp. Math. Statist. Prob.*, Vol. 1 (1967) 281–297.

Rousseeuw, P.J., A visual display for hierarchical classification, in: E. Diday, Y. Escoufier, L. Lebart, J. Pagés, Y. Schektman, and R. Tomassone (Eds.), *Data Analysis and Informatics 4* (North-Holland, Amsterdam, 1986) 743–748.

Rousseeuw, P.J., Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis, *Journal of Computational and Applied Mathematics*, **20** (1987) 53–65.

Rousseeuw, P.J., Discussion: fuzzy clustering at the intersection, *Technometrics*, **37** (1995) 283–286.

Ruspini, E.H., Numerical methods for fuzzy clustering, *Inform. Sci.*, **2** (1970) 319–350.

Statistical Sciences, *S-PLUS Programmer's Manual, Version 3.2* (StatSci, a division of MathSoft, Seattle, 1993).

Williams, W.T. and J.M. Lambert, Multivariate methods in plant ecology, I. Association-analysis in plant communities, *J. Ecology*, **47** (1959) 83–101.

CLOS is intended as an object-oriented environment for the design and development of complex applications with internal parallelism particularly in advanced information systems. 1 Introduction. The object-oriented approach in programming [TDK89] is widely recognized and con-sidered as a promising basis for new software techniques. This approach is relevant to programming languages and systems [AOC+88,Str86], database management systems (DBMS) [KTT89,HK89] and certainly operating systems [ABLN85,YTR+87].Â These connections may be defined in several subsystem components (finally in clusters) since. 426. Struyf, A., Hubert, M. and Rousseeuw, P.J. (1997) Clustering in an Object-Oriented Environment. Journal of Statistical Software, 1, 1-30. has been cited by the following article: TITLE: A New Approach to Investigate Studentsâ€™ Behavior by Using Cluster Analysis as an Unsupervised Methodology in the Field of Education. AUTHORS: Onofrio Rosario Battaglia, Benedetto Di Paola, Claudio Fazio. KEYWORDS: Education, Unsupervised Methods, Hierarchical Clustering, Not-Hierarchical Clustering, Quantitative Analysis.Â Open-Access Framework for Efficient Object-Oriented Development of Video Analysis Software.Â Improvement of Software Quality Attributes in Object Oriented Analysis and Design Phase Using Goal-Question-Metric Paradigm. CHAF - an object-oriented framework for configuring applications in a clustered environment. 9 years 8 months ago. Download blogs.sun.com. - In high availability clustering solutions, an application must be configured properly to run within the framework of the high availability solution. This configuration is often cumbersome, and thus the probability of configuration errors increases. In this paper, we propose a framework, called CHAF, to build configuration tools for high availability clustering solutions. We show how CHAF is scalable to a variety of applications.Â Â» A JavaBased Parallel Programming Support Environment. more Â». Post Info. More Details (n/a). Added. 02 Jun 2010.