Interactive Dendrograms: The R Packages idendro and idendr0

Tomáš Sieger
Czech Technical University in Prague

Catherine B. Hurley
National University of Ireland Maynooth

Karel Fišer
Charles University in Prague

Claudia Beleites
Chemometrix GmbH

Abstract

This introduction to the R packages **idendro** and **idendr0** is based on (Sieger, Hurley, Fišer, and Beleites 2017), published in the Journal of Statistical Software.

Hierarchical cluster analysis is a valuable tool for exploring data by describing their structure using a dendrogram. However, proper visualization and interactive inspection of the dendrogram are needed to unlock the information in the data. We describe a new R package, **idendro**, that enables the user to inspect dendrograms interactively: to select and color clusters, to zoom and pan the dendrogram, and to visualize the clustered data not only in a built-in heat map, but also in any interactive plot implemented in the **cranvas** package. A lightweight version **idendr0** with reduced dependencies is also available from the Comprehensive R Archive Network.

Keywords: interactive graphics, visualization, cluster analysis, exploratory data analysis, high-dimensional data, R.

1. Introduction

Modern experiments often produce moderate- or high-dimensional data. These data can be challenging to explore, as one usually needs to consider information from all dimensions simultaneously. *Hierarchical clustering analysis* (HCA; for an overview, see Hastie, Tibshirani, and Friedman 2009, Section 14.3.12) is a valuable tool that can give an insight into the structure of such data. In brief, HCA tries to reveal the structure of data by modeling it in terms of a hierarchy of clusters of observations. HCA can follow an agglomerative (bottom up) approach, or a divisive (top down) approach. In the agglomerative approach, HCA initially considers each observation to form an elementary cluster, and then builds a hierarchy of clusters by iteratively merging the two most similar clusters into a new one, until there is just a single cluster comprising all the observations. Following the divisive approach, HCA starts from the cluster of all observations, and builds a hierarchy by iteratively splitting each cluster of at least two observations into two clusters. In both cases, HCA results in a hierarchy of clusters, which can be represented by a tree-like structure called *a dendrogram*. Given that we performed agglomerative (or divisive) HCA over n observations, the dendrogram consists

of $n-1$ pairs of branches representing the $n-1$ merge (or split) operations. The height of each pair of branches represents the distance of the two subclusters merged (split) at each step. Without loss of generality, and in order to simplify the text, in the rest of this paper we assume that agglomerative clustering was used.

A comprehensive study of the structure of data requires proper visualization and interactive inspection of the dendrogram. While plotting the whole dendrogram presents the overall structure of the data, any finer structure becomes visible only when focused on, i.e., by zooming into the dendrogram. Moreover, inspecting the dendrogram alone cannot tell which observations form particular clusters. Decorating elementary clusters (i.e., observations) with their labels can help. However, we may still want to know what the values of individual features of observations in particular clusters are. This can be resolved by plotting feature space projections of the data (e.g., a 2D marginal distribution of the data displayed in a scatter plot[1], or using principal component analysis or a projection pursuit tour[2].) and linking them to the dendrogram.

While HCA can be performed easily in R (R Core Team 2016), e.g., using the `hclust` function in the **stats** package (part of R), or the `agnes` or `diana` functions in the **cluster** package (Maechler, Rousseeuw, Struyf, Hubert, and Hornik 2016); for an overview, see Leisch and Grün (2016), truly interactive dendrogram visualization tools are missing. The low-level dendrogram plotting and interaction functions consisting of the `plot` methods for '`dendrogram`' and '`hclust`' objects and the `identify` method for '`hclust`' objects are in the **stats** package, which, however, can satisfy basic needs only, as they offer limited interactivity.

This paper describes the **idendro** package for R (Sieger 2017b), an interactive dendrogram visualization and inspection tool. **idendro** enables the user to plot large dendrograms, which can be zoomed, panned and inspected interactively by selecting and coloring clusters anywhere in the dendrogram. Feature space projections of the data can be visualized in a built-in heat map, or also in any interactive plot implemented in the **cranvas** package (Xie *et al.* 2013, e.g., a scatter plot, or a parallel coordinate plot). Such plots can be used to visualize the data, or to highlight observations forming selected clusters, or the user can also select (*brush*) observations there and then look back in the dendrogram what clusters contain the selected observations.

This text is structured as follows. Installation is covered in Section 2. Section 3 describes `idendro` invocation by way of examples using simple low-dimensional data. The graphical user interface (GUI) of **idendro** is described in Section 4 and its interactivity in Section 5. Section 6 is more technical, and discusses the data structures enabling the interaction between **idendro**, **cranvas**, and the user's code. Finally, Section 7 provides a case study demonstrating how **idendro** can be used to explore flow cytometry data, and Section 8 illustrates the use of **idendro** for exploring high-dimensional spectroscopic data.

[1] As shown in the `idendroWithCranvas` demo.
[2] As shown in the `idendroTour` demo

2. Installation

We decided to implement graphic functionality using the **qtbase** (Lawrence and Sarkar 2015a) and **qtpaint** (Lawrence and Sarkar 2015b) packages based on **Qt**, a cross-platform application and UI framework (**Qt** Project 2015). A comparison with alternative frameworks revealed that the R interfaces to **Qt** were more convenient for implementing fast interactive graphics due to their stability, speed, and rich features, including the ability to build a powerful GUI. In addition, the use of **qtbase** and **qtpaint** enabled seamless integration with interactive plots from the **cranvas** package, which also builds on top of **qtbase** and **qtpaint**. On the other hand, the support for **qtbase** and **qtpaint** was (at the time of writing) limited on the Windows platform (see below).

Prior to installing **idendro**, its prerequisites must be installed, namely the **qtbase**, **qtpaint** and **cranvas** packages. The availability of these packages depended on the operating system that was to be used. On Linux systems, they could be installed quite easily. However, we experienced trouble with NVIDIA Optimus graphics ("errors linking simple shader"), which could be resolved by installing **Bumblebee** (**Bumblebee** Project 2015). On Mac OS X, manual installation was needed. On Windows, demanding manual installation involving getting additional libraries and software and building from sources was essential. Readers are referred to the **idendro** installation instructions at `https://github.com/tsieger/idendro/wiki` to learn details.

Provided you have the **qtbase**, **qtpaint**, and **cranvas** packages installed, you can install **idendro** from `https://github.com/tsieger/idendro` using the **devtools** package (Wickham and Chang 2016):

```
R> devtools::install_github("tsieger/idendro", args = "--no-multiarch")
```

To ease **idendro** adoption for users who have not the packages **qtbase**, **qtpaint**, and **cranvas** installed, we have implemented an alternative lightweight **idendro** version called **idendr0** (Sieger 2017a). It is implemented in terms of base R graphics and a platform-independent Tcl/Tk GUI (made accessible by the **tcltk** package, part of R). Interactive exploration of high-dimensional data is enabled thanks to integration with the **GGobi** dynamic interactive graphics (Swayne *et al.* 2010), made available using the **rggobi** package (Temple Lang, Swayne, Wickham, and Lawrence 2016). The **idendr0** package is available on the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=idendr0`. Information about the latest version of the package can be found online at `https://github.com/tsieger/idendr0`. To install it from within R, run:

```
R> install.packages("idendr0")
```

While **idendr0** implements most of the functionality described in this paper, it lags behind the full version of **idendro** in terms of interactivity and performance. Therefore, users will usually prefer **idendro** over **idendr0**, if it is available on their platform.

A short overview of **idendr0** is given in Appendix A.

3. `idendro` invocation

Let us demonstrate the **idendro** functionality on the *iris* data set (Fisher 1936) available from the **datasets** R package. The data set consists of 150 observations of Iris flowers, 50 observations for each of Setosa, Versicolor, and Virginica species. For each flower, the *sepal length*, *sepal width*, *petal length* and *petal width* were measured (in centimeters) and stored in the first four columns of the *iris* data set. The species indicator (coded as a `factor`) comes in the fifth column.

First, we try to identify clusters (i.e., subgroups of flowers) in the data by performing HCA over the measurements, using the `hclust` function in the **stats** package:

```
R> hc <- hclust(dist(iris[, 1:4]))
```

To visualize `hc`, the resulting hierarchy of clusters represented by a dendrogram, we can simply pass the return value of `hclust`[3] to `idendro`:

```
R> idendro(hc)
```

We get an interactive dendrogram drawn. This can be zoomed and panned, and clusters can be selected in it. However, we cannot see what flowers constitute the individual clusters. We therefore pass the *iris* data set as the second argument to `idendro`, which, by default, enables a heat map to be drawn next to the dendrogram and the names of the observations to be displayed next to the heat map:

```
R> idendro(hc, iris)
```

Now, we get the dendrogram drawn with a heat map attached to it (Figure 1[4]). This plot gives a quite reasonable insight into which observations form which clusters. For example, we can see that the top most cluster (colored in green) includes flowers with short petals.

If we want to visualize data and clusters in other feature space projections, we can make use of any **cranvas** interactive plot, e.g., a scatter plot, or a parallel coordinate plot, by passing the `idendro` return value[5] to the `data` argument of `qscatter` and `qparallel`:

```
R> mdf.iris <- idendro(hc, iris)
R> print(qscatter(Sepal.Length, Sepal.Width, data = mdf.iris))
R> print(qparallel(~ ., data = mdf.iris))
```

Now, we can enjoy the bidirectional integration of **idendro** with **cranvas** (Figure 2). The points in the scatter plot and the parallel coordinate plot get automatically colored according to the currently selected clusters in the dendrogram. Moreover, the points *brushed* in the **cranvas** plots can be directly tracked in a so-called *brushed map* in the dendrogram window, which we describe in the following section.

[3] We could also use return values of other HCA functions, provided they are convertible to class '`hclust`' by the `as.hclust` function. Also, we could optimize the dendrogram using the `dser` function from **DendSer** (Hurley and Earle 2013), as shown in the `idendroDendSer` demo.

[4] Which will be discussed in full detail in Section 4.

[5] This return value holds a so-called *mutable data frame*, the original *iris* data frame enriched with special hidden attributes over which the dendrogram and the other interactive plot can communicate. See Section 6 to learn more.

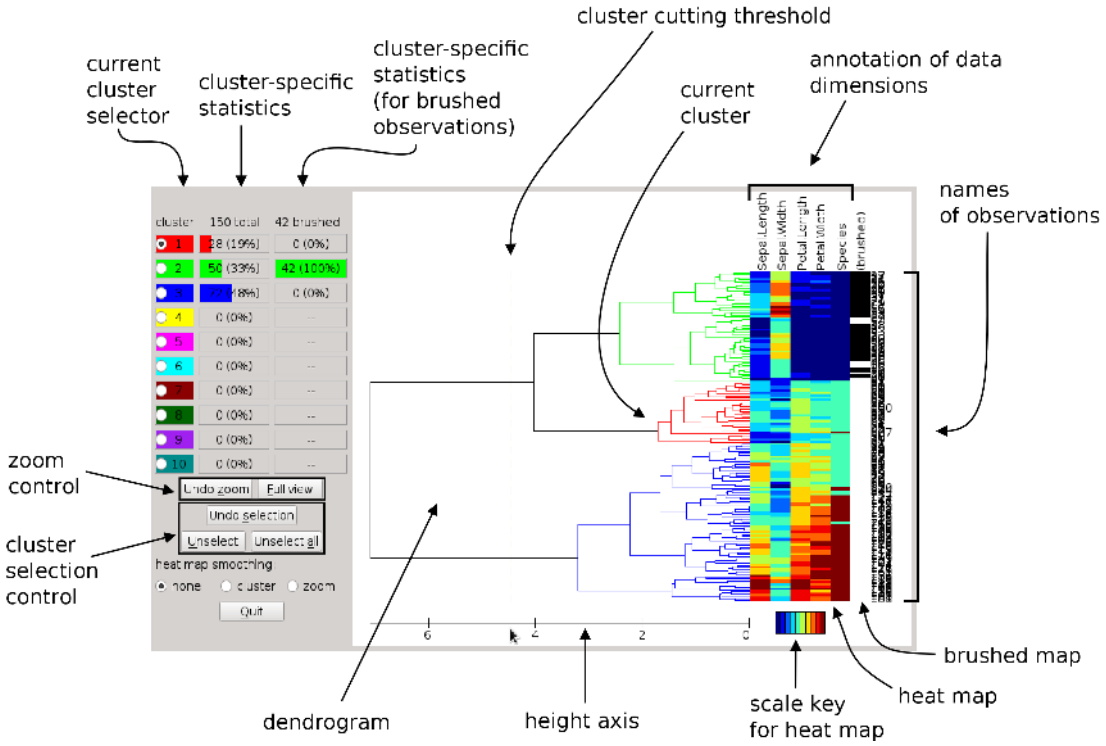

Figure 1: `idendro` window displaying *iris* data in terms of a dendrogram and a heat map.

4. `idendro` window description

The code given above should result in the plot shown in Figure 1. We can see a window consisting of two main components: a simple GUI on the left side and a dendrogram enriched with a heat map and a brushed map on the right side.

The top part of the GUI is populated by three columns:

- the current cluster selector, a radio button group determining which cluster is the *current cluster*. The *current cluster* determines which color and ID will be associated with a cluster selected in the dendrogram,
- cluster-specific statistics telling how many observations out of the total number of observations fall into each cluster, and
- cluster-specific statistics telling how many observations out of the observations brushed currently fall into each cluster.

The number of clusters shown in the GUI can be controlled using the `maxClusterCount` argument to `idendro`. The colors of the clusters can be controlled using the `clusterColors` argument. In the bottom part of the GUI there are buttons that give access to the zoom and cluster selection history, and radio buttons controlling the heat map smoothing mode.

On the right side, there is a *dendrogram* with a heat map and a brushed map attached to it. The dendrogram depicts the process of agglomerative HCA, in which, initially, there were 150 elementary clusters (individual Iris flowers). Iteratively, at each stage, the two closest

clusters got merged, continuing until there was just a single cluster comprising all the 150 observations. The dendrogram is thus formed by 149 pairs of branches, each pair representing one merge operation. The distance of the two clusters merged at a specific stage is called the *height* of the newly merged cluster, and can be read from the axis below the dendrogram. The first merge operation occurred at height close to 0, while the height of the last (the biggest) cluster is close to 7.

The *heat map*, which is attached to the right side of the dendrogram, consists of a row of five colored rectangles drawn next to each observation. The rectangles code graphically the four measurements made on each Iris flower (i.e., *sepal length*, *sepal width*, *petal length*, *petal width*, as shown above the heat map) and the species of each flower. The values of the measurements and the species of each flower can be read by clicking the heat map. Note that the species were coded as a `factor` in the data set and got converted to a numeric type by `idendro` internally, such that the species can be included in the heat map. The heat map colors are defined using the `heatmapColors` argument, which defaults to a list of 10 colors picked from the blue-green-yellow-red color spectrum, but any color spectrum can be used, e.g., `brewer.pal` from the **RColorBrewer** package (Neuwirth 2014), or `gray.colors`, `rainbow`, `heat.colors`, `terrain.colors`, `topo.colors`, or `cm.colors` from the **grDevices** package (part of R). The heat map appearance is controlled using the `heatmapEnabled` argument, and is enabled by default, provided data was passed to `idendro`. The relative size of the heat map can be controlled using the `heatmapRelSize` argument, which determines how much space is reserved for the heat map out of the space reserved for both the dendrogram and the heat map. The default is 0.2, i.e., the heat map takes 20% and the dendrogram 80% of the space. For example, to use the gray color scale of 25 shades of gray in a heat map enlarged to 50% of space, `idendro` can be invoked by:

```
R> idendro(hc, iris, heatmapColors = gray.colors(25), heatmapRelSize = 0.5)
```

The *brushed map*, displayed immediately next to the heat map, is formed by black/white rectangles, indicating whether the corresponding observation is/is not currently brushed in plots integrated with the dendrogram. The brushed map is enabled, by default, provided data was passed to `idendro`. Brushed map visibility can be controlled using the `brushedmapEnabled` argument.

The *names of individual observations* are displayed on the right side of the `idendro` window. They are unreadable in Figure 1, but will become clear once we zoom into the dendrogram. The appearance of the names of the observation can be controlled using the `observationAnnotationEnabled` argument, which is `TRUE`, by default.

5. Interacting with `idendro`

5.1. Cluster selection

`idendro` enables us to select a few clusters in the dendrogram, label and color them, and provide simple summary statistics for them. Initially, there are no clusters selected in the dendrogram[6]. To select a cluster, we can either click on a cluster in the dendrogram (on their

[6]Unless you pass a *mutable data frame* holding cluster selection metadata to `idendro` – see Section 6 for details.

top-level branch), or *cut* the dendrogram at a specified height.

To *select a cluster* in the dendrogram *manually*, we simply click on the top-level branch of the cluster. The cluster gets colored according to the color of the *current cluster* selected in the GUI, and associated with the ID of the *current cluster*. Initially, the *current cluster* is the first one, which is colored in red, by default. To associate another dendrogram cluster with the *current cluster*, we can simply click on that cluster in the dendrogram, which results in unselecting the previous cluster and selecting the new one. To select some other cluster while keeping the first one selected, we simply change the *current cluster* in the current cluster selector in the GUI and pick another cluster from the dendrogram.

We can also *select clusters by* cutting the dendrogram at a specified height threshold, i.e., select all clusters merged at or below the specified threshold. To *cut* the dendrogram, we move the mouse pointer below the dendrogram axis (this results in displaying the *cutting* threshold across the dendrogram, see Figure 1) and press the left mouse button. `idendro` selects all the clusters merged at or below the specified height, and associates them with the first few clusters in the GUI. This is what we see in Figure 1, in which we *cut* the dendrogram at a height of about 3.7, and obtained three selected clusters (red, green, and blue clusters consisting of 28, 50, and 72 observations, respectively).

We can see that these three selected clusters do not reflect the nature of the data, since there were 50 flowers of each species. However, we can ask to what extent the clusters reflect the natural structure of the data. Luckily, the heat map can help to answer this question. The last column of the heat map shows the species of individual flowers, coded numerically (coerced from the levels of the *Species* factor) and printed in text when we click the heatmap. We can see that the second cluster (as shown in GUI, colored in green), which consists of 50 flowers, matches the Iris Setosa flowers perfectly. The first cluster (colored in red) consists of 27 Iris Versicolor flowers and 1 Iris Virginica flower. The third cluster, however, seems to be a mixture of Versicolor and Virginica flowers, though the subclusters of this cluster seem to reflect the structure of the data quite well.

To *unselect* the *current cluster*, i.e., to dissociate the *current cluster* shown in GUI from any cluster shown in the dendrogram, we can click the `"Unselect"` button. The `"Unselect all"` button can be used to unselect all clusters. The selection history is available – the previous selection can be recalled using the `"Undo selection"` button.

5.2. Zooming and panning

Dendrogram inspection usually involves iteratively focusing on clusters at different heights in different parts of the dendrogram. For example, we might want to study the internal structure of one of a few top-level clusters, taking a deeper and deeper look into it iteratively. `idendro` enables such inspection by zooming and panning the dendrogram.

To *zoom* in the dendrogram, we can either define a region to zoom to explicitly, using right mouse click and drag, or using the mouse wheel. In the latter case, the amount of zoom can be controlled using the `zoomFactor` argument.

To restore the original dendrogram view (i.e., to zoom out maximally), we can click the `"Full view"` button. The zoom history can be recalled by clicking the `"Undo zoom"` button.

The dendrogram can be *panned* using mouse drag.

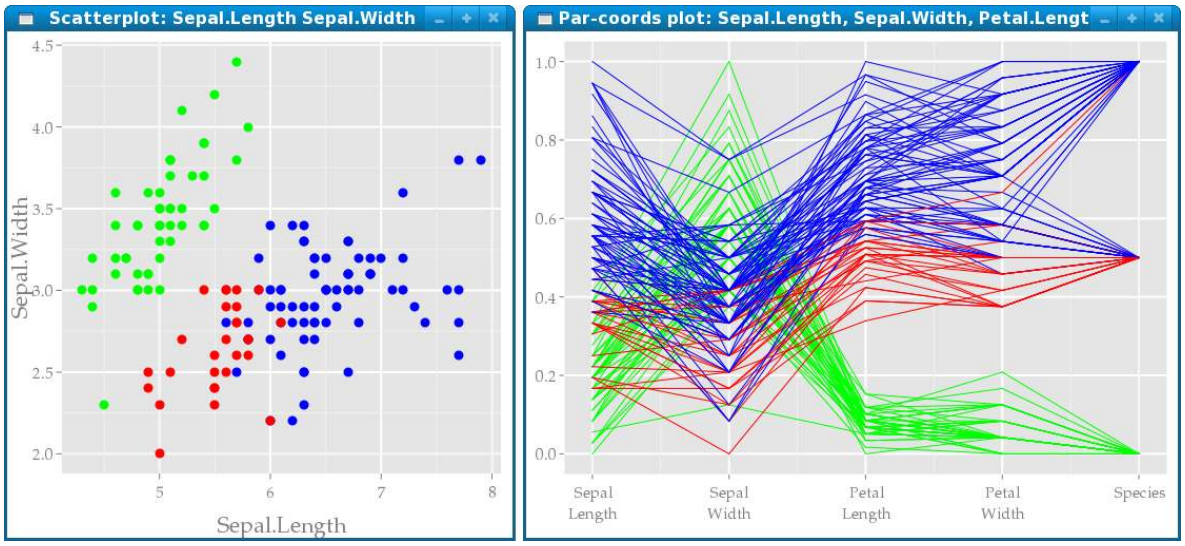

Figure 2: Interactive **cranvas** plots integrated with **idendro**. The scatter plot (left) and the parallel coordinate plot (right) display measures made on the Iris flowers, reflecting the color of clusters selected in the dendrogram (Figure 1).

6. Mutaframes: Data structures for dynamic integration

In this section we describe *mutable data frames*, or *mutaframes* – the data structure provided by the **plumbr** package (Lawrence and Wickham 2014) that enables the integration of interactive plots of **idendro** and **cranvas** with each other, and also with the user's code.

6.1. Integration with other interactive plots

We keep in mind that it hardly suffices to look at a dendrogram and a heat map alone to learn what data tells us. We usually need to explore more feature space projections of the data. Hopefully, thanks to the effort made by the authors of the **cranvas** package, **idendro** can be bidirectionally integrated with modern high-speed interactive **cranvas** plots (Figure 2). **idendro** automatically highlights observations forming the currently selected clusters in these plots. Moreover, selecting (*brushing*) observations in these plots propagates instantly into the brushed map, which then shows what clusters contain the selected observations (Figure 1).

Technically speaking, the integration of **idendro** with **cranvas** interactive plots is enabled thanks to the concept of *mutable data frames* implemented in the **plumbr** package. In brief, *mutable data frames* are data frames enriched with hidden metadata (special columns in the data frame) that can be read, written and applications/users can also listen to changes being made to them. **cranvas** defines metadata controlling the color (`.color`, `.border`), size (`.size`) and visibility (`.visible`) of observations in plots. It also defines the `.brushed` metadata, which controls whether a given observation is brushed currently.

Mutable data frames can be explicitly constructed using the `qdata` function in **cranvas**:

```
R> mdf.iris <- qdata(iris)
```

Alternatively, `idendro` converts the data it gets into a *mutable data frame* automatically and returns it, such that you can get a mutable data frame as a side-effect of `idendro` invocation:

```
R> mdf.iris <- idendro(hc, iris)
```

`idendro` alters the `.color` and `.border` metadata to color observations according to the color of the clusters they appear in, and listens to changes being made to the `.brushed` metadata to learn what observations are currently being brushed.

6.2. Persisting cluster selection

When inspecting the dendrogram, we may wish to persist the clusters found so far, such that we will be able to get back to them later.

idendro introduces two more *mutable data frame* metadata in addition to those defined by **cranvas**: `.cluster` and `.inCurrentCluster`. For each observation, the `.cluster` holds the ID of the cluster that the observation is a member of (or 0, if the observation does not belong to any cluster). Similarly, the `.inCurrentCluster` metadata determines whether the given observation is a member of the *current cluster*.

The `.cluster` metadata can be used to persist the selected clusters in the dendrogram by simply saving the *mutable data frame* returned by `idendro`:

```
R> mdf.iris <- idendro(hc, iris)
```

To recall the persisted clusters, we can invoke `idendro` passing the saved *mutable data frame* as its second argument:

```
R> idendro(hc, mdf.iris)
```

Note, however, that the selection history cannot be persisted in this way.

The `.inCurrentCluster` metadata can be used by the user's code to compute information specific to the *current cluster* set in the GUI, as shown in the `idendroWithUserCallback.R` demo, in which we install a listener on the `mdf.iris` *mutaframe* and print the number and the mean *sepal length* of the observations in the current cluster whenever the cluster changes:

```
R> mdf.iris <- idendro(hc, iris)
R> my.listener <- add_listener(mdf.iris, function(i, j) {
+    if (".inCurrentCluster" %in% j) {
+      cat(sprintf(
+        "The current cluster consists of %d observation(s).
+        The mean sepal length is %.3f.\n",
+        sum(mdf.iris$.inCurrentCluster),
+        mean(mdf.iris$Sepal.Length[mdf.iris$.inCurrentCluster])))
+    }
+  })
```

Note that the listener can be removed when not needed by:

```
R> remove_listener(mdf.iris, my.listener)
```

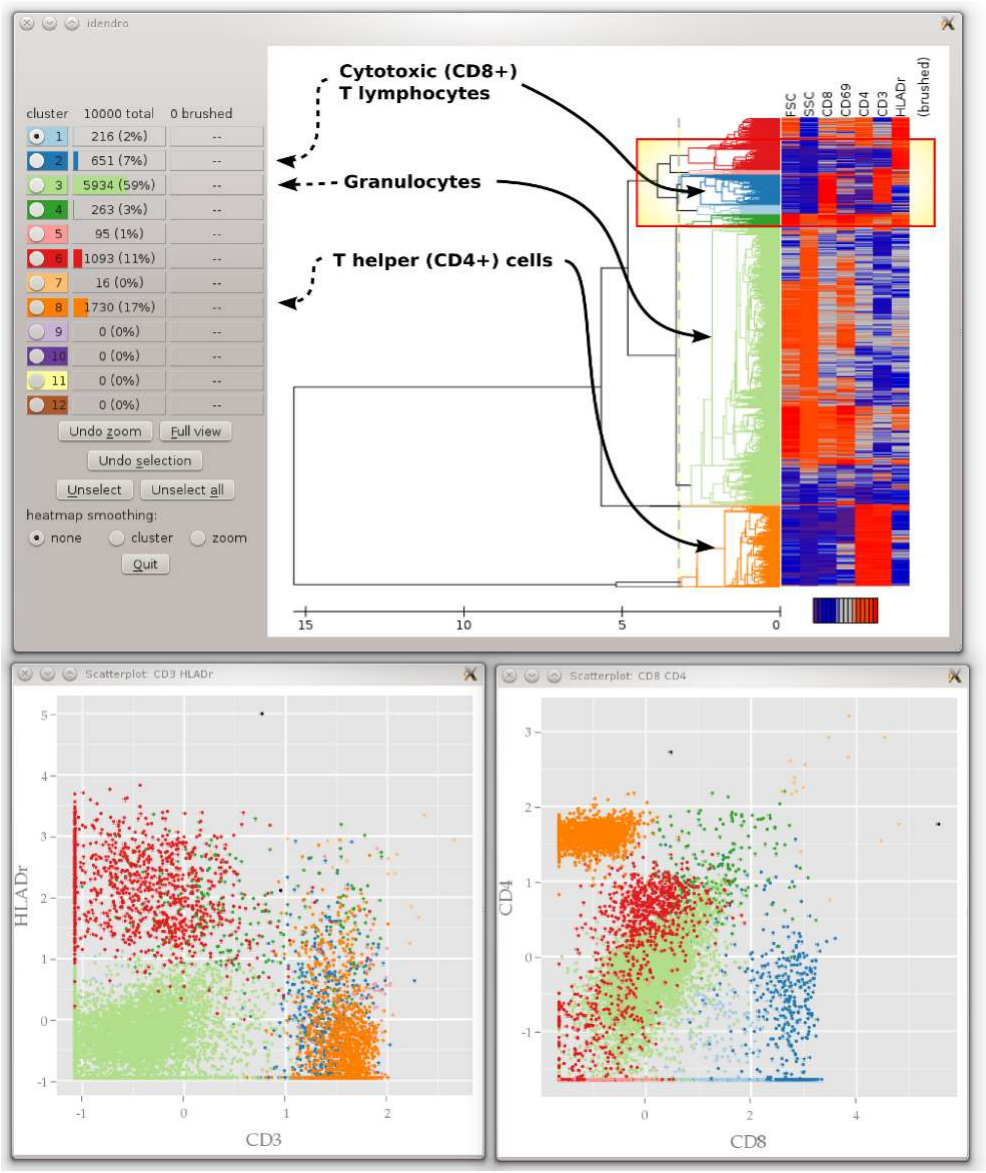

Figure 3: `idendro` window and two scatter plots of HCA of flow cytometry data. Clusters were selected automatically by threshold cutting, and correspond with biologically relevant cell populations. The red rectangle marks the area zoomed into in Figure 4.

7. Case study: Exploration of flow cytometry data

Flow cytometry is a tool for analyzing live cells in a wide range of biomedical settings (Brown and Wittwer 2000). Flow cytometry measures multiple parameters on thousands to millions of individual cells in a single experiment. Such experiments produce large data sets, which are difficult to explore, partly because researchers typically need to explore and interpret all the measured parameters. While traditionally the data are analyzed manually by drawing regions of interest on two-parameter scatter plots, novel approaches are emerging. One of them is HCA (Fišer *et al.* 2012).

The use of **idendro** provides excellent interplay between traditional analysis and HCA of flow cytometry data. **idendro** facilitates display of all data points and all measured parameters in a heat map, which itself provides novel insights into flow cytometry data. **idendro** also displays cellular hierarchy as computed by HCA in the form of a dendrogram. In addition, **idendro** allows traditional plotting of the data on scatter plots. The strongest point here is the interactivity of **idendro**: Cluster selection on a dendrogram propagates to scatter plots, and scatter plot highlighting (*brushing*) can also be displayed on the side of the heat map. This level of interactivity greatly enhances the ability to compare traditional analysis to clustering. Moreover, the **idendro** window also readily displays the proportions of the data points (here cells) selected by both methods.

Another common aspect of the two analytical methods is the need to fine-tune the cluster selection. This is again facilitated by the **idendro** interface, which enables deselection of clusters and several levels of "undo" action.

Similarly, as the number of data points in flow cytometry data sets is in the range of at least thousands, the ability of **idendro** to zoom and pan the dendrogram serves fine and more accurate cluster selection.

An example code of HCA of flow cytometry data is shown below. It makes use of the `ITN` data from the **flowStats** package (Hahne, Gopalakrishnan, Khodabakhshi, Wong, and Lee 2012). First the data gets transformed, and then agglomerative HCA utilizing the "average" method is performed. The **RColowBrewer** package (Neuwirth 2014) is used to generate the cluster colors.

```
R> library("RColorBrewer")
R> library("idendro")
R> library("flowStats")
R> data("ITN", package = "flowStats")
R> x <- exprs(ITN$sample03[, 1:7])
R> x[, 3:7] <- log10(x[, 3:7])
R> x <- scale(x)
R> hx <- hclust(dist(x), method = "average")
R> mdf.x <- idendro(hx, x,
+    heatmapColors = colorRampPalette(c("purple4", "blue3", "blue3", "grey",
+      "grey", "orangered", "orangered", "red"))(15),
+    clusterColors = brewer.pal(12, "Paired"))
R> print(qscatter("CD3", "HLADr", mdf.x))
R> print(qscatter("CD8", "CD4", mdf.x))
```

The code results in plots similar to Figure 3, in which, however, the clustering described in Fišer *et al.* (2012) was used instead of `hclust`. Clusters of (blood) cells are shown both in the dendrogram and in traditional scatter plots. The color of individual cells in the scatter plots reflects the color of the clusters revealed by HCA and selected in the dendrogram. This enables identification of major cell populations right in the dendrogram. The dendrogram can be iteratively zoomed and panned to explore subpopulations of the major populations of cells. This could not be accomplished easily using the means of static non-interactive visualization. The heatmap can also be consulted to help identify subpopulations of cells of interest by giving a quantitative overview of the values of individual parameters measured on them.

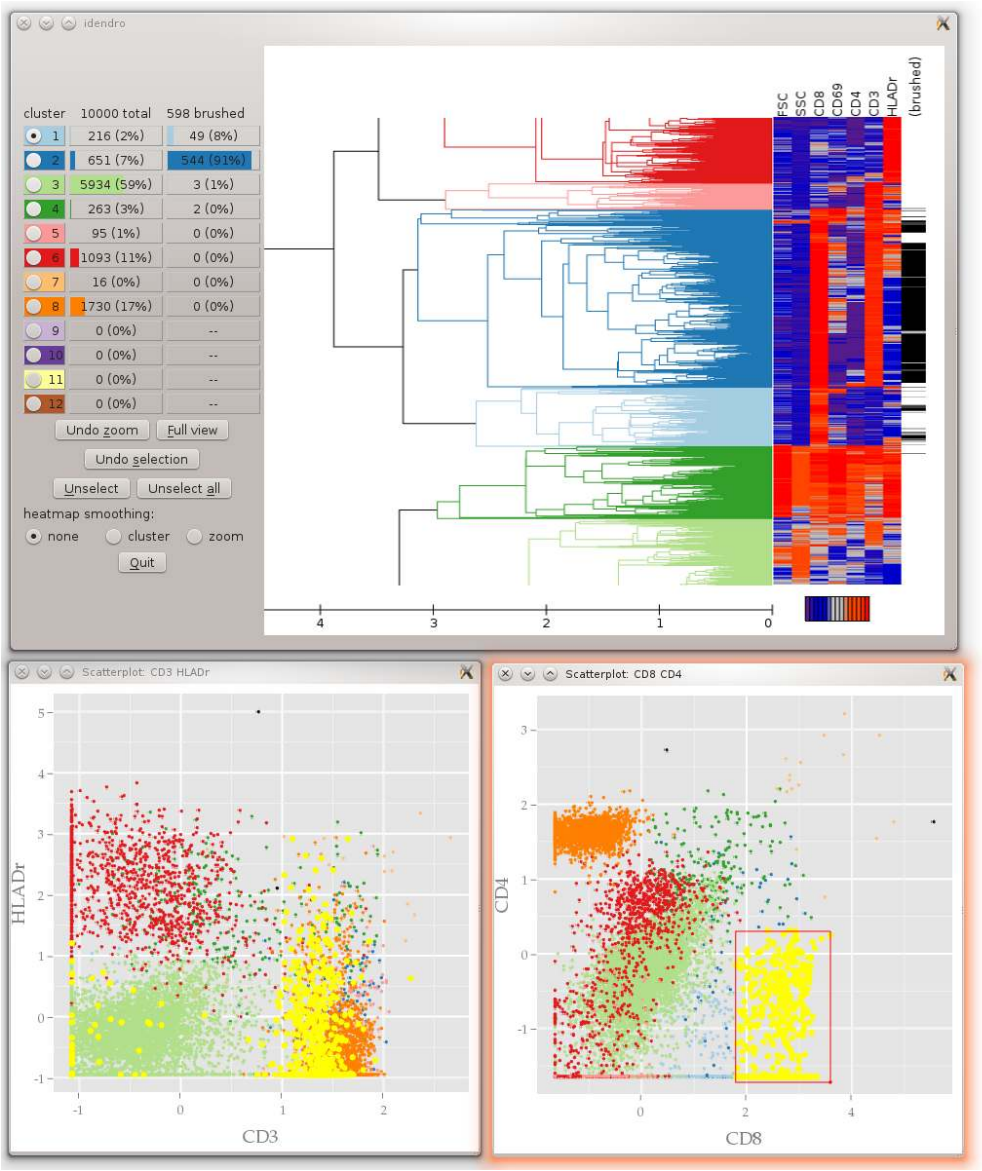

Figure 4: Interaction between **idendro** and **cranvas** plots. Displaying only part of the dendrogram and heatmap from Figure 3 allows finer inspection of the data. The difference between the blue and green clusters (e.g., in CD4 parameter) is clearly visible both in the heat map and in the scatter plots. This can be further verified by brushing cells forming the blue cluster in the lower right scatter plot (yellow points). Brushed cells are also marked in the brushed map on the right side of the heatmap. Statistics of both selections (clusters and brushed points) are given in the main `idendro` window.

Figure 4 shows an example of exploring a smaller cell population. Cells forming the cluster of interest (colored in blue) can be compared to cells selected by traditional brushing cells in scatter plots. Figure 4 shows a good correspondence between HCA and traditional exploration.

8. Case study: idendro and hyperSpec

This example session demonstrates how to use **idendro** together with spectroscopic data stored in 'hyperSpec' objects. First, **idendro** and **hyperSpec** (Beleites and Sergo 2013) are needed:

```
R> library("hyperSpec")
R> library("idendro")
```

The data set we use here is the original version of **hyperSpec**'s chondro data. Briefly, this is a data set of laterally resolved Raman spectra of a cartilage section measured under a microscope. An area of 34×24 µm^2 is covered by a regular grid of 1×1 µm^2. At each grid point, a complete Raman spectrum in the spectral range 600 cm^{-1} to 1800 cm^{-1} was acquired. For further information about the application of micro-Raman spectroscopy to cartilage, see e.g., Bonifacio *et al.* (2010). More information about the data set, and also a more thorough discussion of the pre-processing steps, are available in **hyperSpec**'s "chondro" vignette. The vignette source, together with the original raw data, is available as http://hyperspec.R-Forge.R-project.org/blob/chondro.zip. However, this example session can also be followed with the compressed version shipped with **hyperSpec** (this yields slightly different clustering).

```
R> if (file.exists("chondro.txt")) {
+    chondro <- scan.txt.Renishaw("chondro.txt", data = "xyspc")
+  }
```

In order to obtain a meaningful clustering, baseline correction and normalization is necessary. In addition, we trade spectral resolution for a better signal-to-noise ratio and perform a smoothing interpolation onto an evenly spaced wavenumber axis:

```
R> chondro <- spc.loess(chondro, newx = seq (602, 1800, by = 4))
R> chondro <- chondro - spc.fit.poly.below(chondro)
R> chondro <- sweep(chondro, 1, rowMeans(chondro), "/")
```

The spectra are now extremely similar. In order to emphasize the differences between the spectra, we can subtract the spectrum of the common composition of the whole sample. In theory, this should be the minimum observed intensity at each wavenumber. As the minimum tends to "collect" noise, we use the 5th percentile instead.

```
R> overall.composition <- quantile(chondro, 0.05)
R> chondro <- sweep(chondro, 2, overall.composition, "-")
```

Now the data is ready for HCA. For intensity normalized spectroscopic data, Euclidean distance and Ward's method for fusion of the clusters are often a good choice:

```
R> dst <- dist(chondro)
R> dndr <- hclust(dst, method = "ward")
```

8.1. Linked idendro and cranvas plots

An `idendro` interactive dendrogram can be connected with other plots that use the same mutaframe, such as `qscatter` or `qparallel` plots (available from package **cranvas**).

A suitable mutaframe can be built from the wide-format representation of the 'hyperSpec' object. Note that this does not need to be the exact representation that was used for the HCA, as long as the rows correspond one to one. Therefore, we could have built the mutaframe from the original spectra, including the matrix composition, while the dendrogram was calculated after subtraction. However, for the clarity of the presentation here, we display the difference spectra that were actually used for the cluster analysis:

```
R> mdf.chondro <- as.wide.df(chondro)
```

In order to help qparallel produce readable axis labels, we provide labels every 50 wavenumbers:

```
R> colnames(mdf.chondro)[-(1:3)] <-
+    paste("wl", colnames(mdf.chondro)[-(1:3)], sep = ".")
R> names <- as.character(wl(chondro))
R> names[wl(chondro) %% 50 != 0] <- ""
```

Now the data frame is ready to be converted into a mutaframe:

```
R> mdf.chondro <- qdata(mdf.chondro)
```

and finally, the connected interactive plots can be generated (Figure 5):

```
R> idendro(dndr, mdf.chondro, heatmapRelSize = 0.75,
+    heatmapColors = alois.palette(25))
R> print(qscatter(x, y, data = mdf.chondro, unibrushcolor = FALSE))
R> print(qparallel(vars = var_names(~., mdf.chondro)[-(1:3)],
+    data = mdf.chondro, names = names, scale = "I", glyph = "line"))
```

8.2. User-defined callback functions

hyperSpec already defines a number of sophisticated, though not interactive, plotting methods. To use these plots with idendro, a callback function can be defined that updates these plots whenever the cluster selection changes:

```
R> dev.new()
R> dev.map <- dev.cur()
R> par(mar = c(4, 4, 0.5, 0.5))
R> dev.new()
R> dev.spc <- dev.cur()
R> par(mar = c(4, 4, 0.5, 0.5))
R> spc.callback <- function(i, j) {
+    if (j == ".cluster") {
+      cluster.levels <- ! duplicated(mdf.chondro$.cluster)
+      clusters <- mdf.chondro$.cluster[cluster.levels]
+      cols <- c("black", rep(NA, max(clusters)))
+      cols[clusters + 1] <- mdf.chondro$.color[cluster.levels]
```

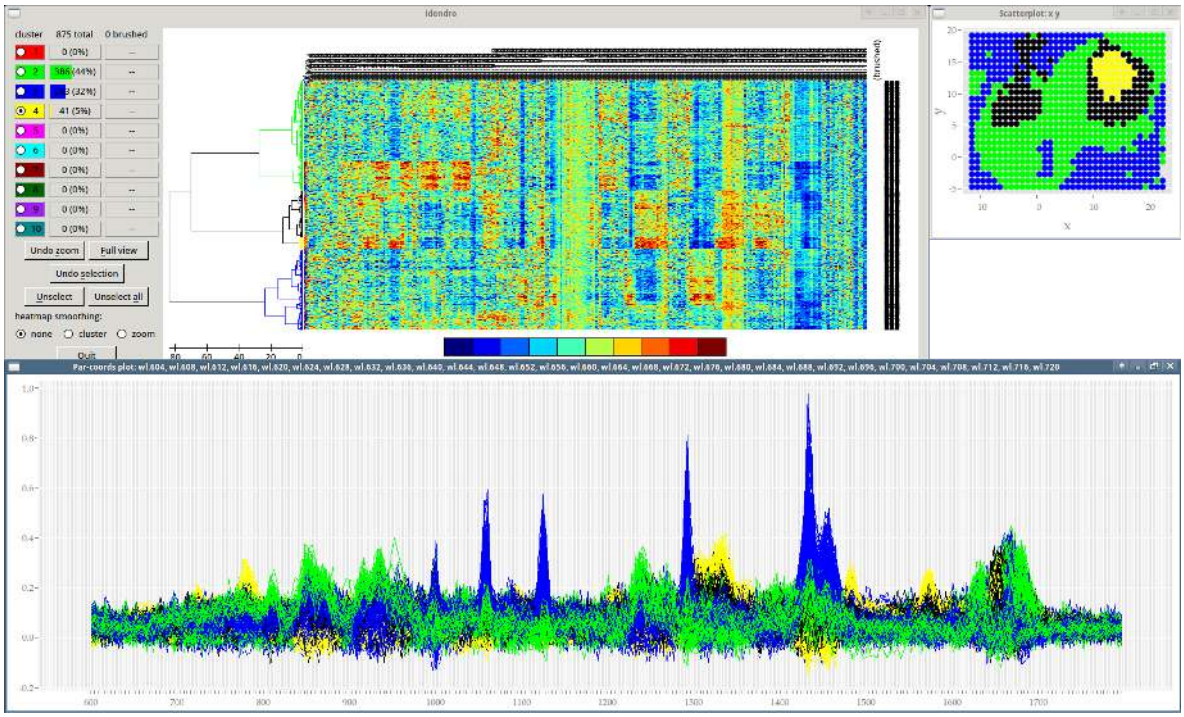

Figure 5: `idendro` linked to interactive plots from **cranvas**: The spectra in the lower window (`qparallel`) are colored according to the clustering selected in the `idendro` window. The spatial distribution of the clusters is shown as a `qscatter` plot. All three plots are linked, so interacting with one will update the display of the others.

```
+
+       dev.set(dev.map)
+       chondro$.cluster <- factor(mdf.chondro$.cluster,
+         levels = 0:max(clusters))
+       print(plotmap(chondro, .cluster ~ x * y, col.regions = cols))
+
+       dev.set(dev.spc)
+       tmp <- aggregate(chondro, chondro$.cluster,
+         quantile, c(0.16, .5, .84))
+       plotspc(tmp, stacked = ".aggregate", fill = ".aggregate",
+         col = cols[sort(unique(mdf.chondro$.cluster)) + 1])
+     }
+   }
```

Again, we construct the mutaframe from the `chondro` object. However, for the plotting of the spectra, our callback function rather uses the 'hyperSpec' object `chondro`. The mutaframe can therefore omit the actual spectra, and use only the so-called extra-information (x and y coordinates, see **hyperSpec**'s "introduction" vignette) of `chondro`:

```
R> mdf.chondro <- qdata(chondro$..)
R> idendro(dndr, mdf.chondro, heatmapEnabled = FALSE)
R> l.map <- add_listener(mdf.chondro, spc.callback)
```

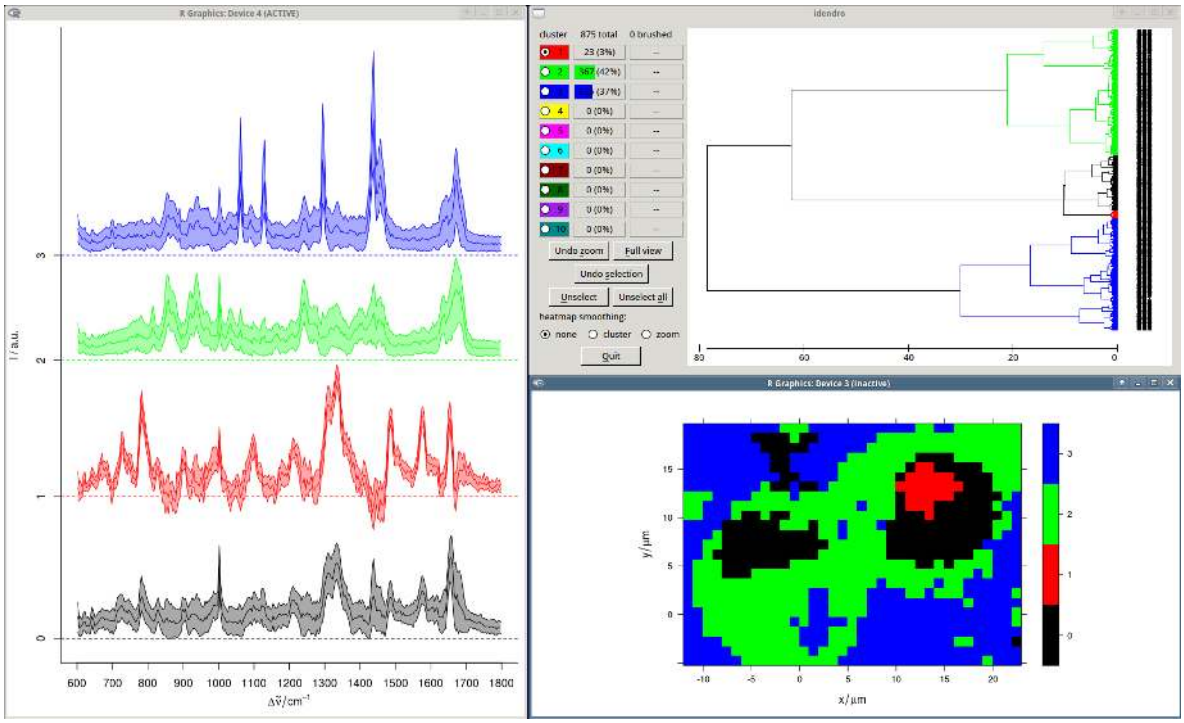

Figure 6: `idendro` together with static graphical information about the displayed clusters. Here, only the `idendro` window provides interaction, and the spectra and map view update accordingly.

Figure 6 shows this approach.

9. Conclusion

idendro, a new R package enabling interactive dendrogram visualization and exploration, has been introduced. To our knowledge, this is the first package enabling really interactive exploration of large dendrograms in R. Moreover, the integration with interactive plots provided by the **cranvas** package makes **idendro** a general data exploration tool.

As the packages that **idendro** depends on were not fully supported on the Windows platform at the time of writing, we have implemented a cross-platform version of **idendro**, called **idendr0**, which provides mostly the same functionality as is provided by **idendro**. To serve the purpose of a general data exploration tool, **idendr0** integrates with interactive plots provided by **GGobi**, a predecessor of the **cranvas** package.

Contributions

The **idendro** package was written by TS. The development started as a Google Summer of Code 2012 (Google Inc. 2012) project, mentored by CH and CB. CB also helped to shape the project before it had started. KF outlined the goals of the project, tested and provided feedback. This paper was drafted by TS, KF (flow cytometry case study) and CB (spectroscopic case study), and was critically reviewed by all authors.

Acknowledgements

The initial development of the **idendro** package was supported by Google under the Google Summer of Code 2012 (Google Inc. 2012). Later development and writing of this paper was supported by grant IGA NT 14387 and grant IGA NT 13462 from Ministry of Health, Czech Republic, and the European social fund within the framework of realizing the project "Support of inter-sectoral mobility and quality enhancement of research teams at Czech Technical University in Prague", CZ.1.07/2.3.00/30.0034. We are grateful to Jiří Wild for testing the installation on Mac OS.

References

Beleites C, Sergo V (2013). ***hyperSpec**: A Package to Handle Hyperspectral Data Sets in R*. R package version 0.98-20130516, URL http://hyperspec.R-Forge.R-project.org.

Bonifacio A, Beleites C, Vittur F, Marsich E, Semeraro S, Paoletti S, Sergo V (2010). "Chemical Imaging of Articular Cartilage Sections with Raman Mapping, Employing Uni- And Multi-Variate Methods for Data Analysis." *The Analyst*, **135**(12), 3193–3204. doi:10.1039/c0an00459f.

Brown M, Wittwer C (2000). "Flow Cytometry: Principles and Clinical Applications in Hematology." *Clinical Chemistry*, **46**(8 Pt 2), 1221–1229.

Bumblebee Project (2015). ***Bumblebee**: A Project Aiming to Support NVIDIA Optimus Technology under Linux*. URL http://bumblebee-project.org/.

Fisher RA (1936). "The Use of Multiple Measurements in Axonomic Problems." *The Annals of Eugenics*, **7**(2), 179–188. doi:10.1111/j.1469-1809.1936.tb02137.x.

Fišer K, Sieger T, Schumich A, Wood B, Irving J, Mejstříková E, Dworzak MN (2012). "Detection and Monitoring of Normal and Leukemic Cell Populations with Hierarchical Clustering of Flow Cytometry Data." *Cytometry A*, **81**(1), 25–34. doi:10.1002/cyto.a.21148.

Google Inc (2012). *Google Summer of Code 2012*. URL http://www.google-melange.com/gsoc/homepage/google/gsoc2012.

Hahne F, Gopalakrishnan N, Khodabakhshi AH, Wong CJ, Lee K (2012). ***flowStats**: Statistical Methods for the Analysis of Flow Cytometry Data*. R package version 1.16.0, URL http://www.bioconductor.org/packages/2.12/bioc/html/flowStats.html.

Hastie T, Tibshirani R, Friedman J (2009). *The Elements of Statistical Learning*. 2nd edition. Springer-Verlag. doi:10.1007/978-0-387-84858-7.

Hurley CB, Earle D (2013). ***DendSer**: Dendrogram Seriation: Ordering for Visualisation*. R package version 1.0.1, URL https://CRAN.R-project.org/package=DendSer.

Lawrence M, Sarkar D (2015a). ***qtbase**: Interface between R and **Qt***. R package version 1.0.11, URL https://CRAN.R-project.org/package=qtbase.

Lawrence M, Sarkar D (2015b). ***qtpaint**: **Qt**-Based Painting Infrastructure*. R package version 0.9.1, URL https://CRAN.R-project.org/package=qtpaint.

Lawrence M, Wickham H (2014). ***plumbr**: Mutable and Dynamic Data Models*. R package version 0.6.9, URL https://CRAN.R-project.org/package=plumbr.

Leisch F, Grün B (2016). *CRAN Task View: Cluster Analysis & Finite Mixture Models*. Version 2016-11-24, URL https://CRAN.R-project.org/view=Cluster.

Maechler M, Rousseeuw P, Struyf A, Hubert M, Hornik K (2016). ***cluster**: Cluster Analysis Basics and Extensions*. R package version 2.0.5, URL https://CRAN.R-project.org/package=cluster.

Neuwirth E (2014). ***RColorBrewer**: ColorBrewer Palettes*. R package version 1.1-2, URL https://CRAN.R-project.org/package=RColorBrewer.

Qt Project (2015). ***Qt** Application Framework*. URL http://www.qt.io/developers/.

R Core Team (2016). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Sieger T (2017a). ***idendr0**: Interactive Dendrograms*. R package version 1.5.3, URL https://CRAN.R-project.org/package=idendr0.

Sieger T (2017b). ***idendro**: Interactive Dendrograms*. R package version 1.5.3, URL https://github.com/tsieger/idendro.

Sieger T, Hurley CB, Fišer K, Beleites C (2017). "Interactive Dendrograms: The R Packages idendro and idendr0." *Journal of Statistical Software*, **76**(10), 1–22. doi:10.18637/jss.v076.i10.

Swayne D, Cook D, Temple Lang D, Buja A, Lewin-Koh N, Hofmann H, Lawrence M, Wickham H (2010). ***GGobi**, an Open Source Visualization Program for Exploring High-Dimensional Data*. URL http://ggobi.org/.

Temple Lang D, Swayne D, Wickham H, Lawrence M (2016). ***rggobi**: Interface between R and **GGobi***. R package version 2.1.21, URL https://CRAN.R-project.org/package=rggobi.

Wickham H, Chang W (2016). ***devtools**: Tools to Make Developing R Code Easier*. R package version 1.12.0, URL https://CRAN.R-project.org/package=devtools.

Xie Y, Hofmann H, Cook D, Cheng X, Schloerke B, Vendettuoli M, Yin T, Wickham H, Lawrence M (2013). ***cranvas**: Interactive Statistical Graphics Based on **Qt***. R package version 0.8.2, URL http://github.com/ggobi/cranvas.

A. idendr0 overview

This appendix gives an overview of **idendr0**, a cross-platform backport of the **idendro** package, implemented using base R graphics embedded in a **Tcl/Tk** GUI. Essentially, the usage and functionality of **idendr0** is the same as the usage and functionality of **idendro**: They both provide the `idendro` function, taking mostly the same parameters.

As in Section 3, we demonstrate the **idendr0** functionality on the *iris* data set.

First, we identify clusters of Iris flowers using HCA:

```
R> hc <- hclust(dist(iris[, 1:4]))
```

and visualize the resulting hierarchy of clusters by passing it to `idendro`:

```
R> library("idendr0")
R> idendro(hc)
```

We get an interactive dendrogram drawn, which can be zoomed and panned, and clusters can be selected in it. By passing the *iris* data set as the second argument to `idendro`, we enable a heat map to be drawn next to the dendrogram (Figure 7, right):

```
R> idendro(hc, iris)
```

If we want to visualize data and clusters in other feature space projections, we can integrate `idendro` with simple R plots, as shown in the `idendroWithScatter` demo:

```
R> plot(iris$Sepal.Length, iris$Sepal.Width, pch = 19)
R> colorizeCallback <- function(clr) {
+    clusterColors <- c('black', 'red', 'green', 'blue', 'yellow', 'magenta',
+        'cyan', 'darkred', 'darkgreen', 'purple', 'darkcyan')
+    plot(iris$Sepal.Length, iris$Sepal.Width,
+        col = clusterColors[clr + 1], pch = 19)
+  }
R> idendro(hc, iris, colorizeCallback = colorizeCallback)
```

In this example, we install a callback function that is called whenever we select (and thus color) some cluster in the dendrogram, such that the color of the observations shown in the scatter plot reflects the color of the cluster that each observation is a part of. Similarly, the `idendroWithScatterAndParcoord` demo integrates with both a scatter plot and a parallel coordinate plot, as shown in Figure 7.

Additionally, we can easily integrate `idendro` with **GGobi** by setting the `ggobi` argument to `TRUE` (and, optionally, overriding the default **GGobi** glyph type and size), as shown in the `idendroWithGgobi` demo:

```
R> idendro(hc, iris, ggobi = TRUE, ggobiGlyphType = 4, ggobiGlyphSize = 3)
```

This results in starting an instance of **GGobi**, which produces, by default, a scatter plot matrix of all the 2D projections of the supplied Iris data[7] (Figure 8), and the installation of two

[7] Of course, we can create more **GGobi** plots later.

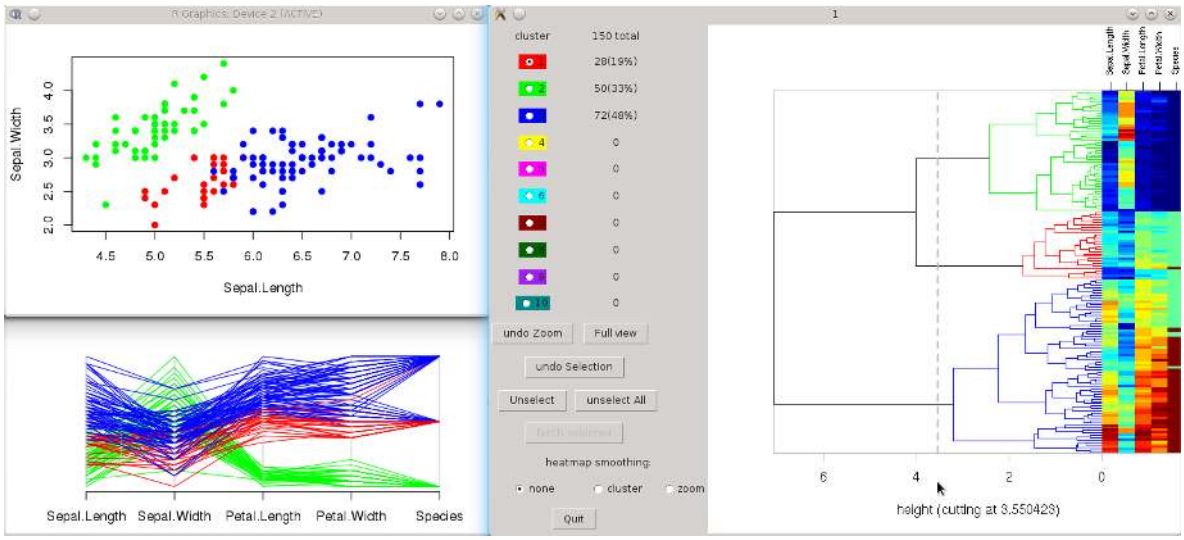

Figure 7: A scatter plot and a parallel coordinate plot as created using base R graphics and integrated with **idendr0**. The color of the observations in the plots reflects the color of the clusters in the dendrogram. For a comparison with similar plots created by Qt-based **idendro**, see Figures 1 and 2.

Figure 8: Interactive **GGobi** scatter plot matrix integrated with **idendr0**. The color of observations in the scatter plot matrix instantly reflects the currently selected clusters in the dendrogram, while the brushed map in the dendrogram window can show which observations are currently selected in **GGobi**. Note that the heat map smoothing mode is set to `cluster`, such that the heat map displays the cluster-specific means instead of individual measurements.

callbacks that ensure bidirectional integration of `idendro` with **GGobi**. `colorizeCallback` is called automatically to instantly color observations in **GGobi** according to the currently selected clusters in the dendrogram. `fetchSelectedCallback` serves the opposite purpose: It can be used to track observations selected (brushed) in **GGobi** currently, and display this in the `idendro` brushed map. `fetchSelectedCallback` is called explicitly, by pressing the

`"fetCh selected"` button[8]. (Here we can see one limitation of **idendr0** – while **idendro** is able to update the brushed map automatically, in **idendr0**, we need to update the brushed map explicitly.)

In **idendr0**, we can also persist the cluster selection that has been made: `idendro` returns a numeric vector of cluster membership of each observation, in which `0`s denote unselected observations, and values of `i > 0` denote observations forming the `i`th cluster. If we later pass this vector as the `clusters` argument to `idendro`, we recall the persisted cluster selection, as shown in the `idendroPersistent` demo:

```
R> clusters <- idendro(hc, iris)
R> idendro(hc, iris, clusters = clusters)
```

To learn more, readers are referred to demos in the **idendr0** package. Notably, the `idendroFlow` and `idendroChondro` demos mirror the two use cases presented in Sections 7 and 8.

Affiliation:

Tomáš Sieger
Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University in Prague
and
Department of Neurology and Center of Clinical Neuroscience
1st Faculty of Medicine and General University Hospital, Charles University in Prague
Prague, Czech Republic
E-mail: tomas.sieger@seznam.cz

Catherine B. Hurley
National University of Ireland Maynooth
Maynooth, Ireland
E-mail: catherine.hurley@nuim.ie

Karel Fišer
CLIP – Childhood Leukaemia Investigation Prague
Department of Paediatric Haematology and Oncology
2nd Faculty of Medicine, Charles University in Prague
and
University Hospital Motol
Prague, Czech Republic
E-mail: karel.fiser@lfmotol.cuni.cz

Claudia Beleites
Chemometric Consulting and Chemometrix GmbH
Södeler Weg 19, 61200 Wölfersheim, Germany
E-mail: Claudia.Beleites@chemometrix.eu

[8]The capital 'C' refers to the shortcut key assigned to this button.

and
Department of Spectroscopy and Imaging, Leibniz-Institute of Photonic Technology
Albert-Einstein-Str. 9, 07745 Jena, Germany