TRAMPR: A package for analysis of Terminal-Restriction Fragment Length Polymorphism (TRFLP) data

Rich FitzJohn & Ian Dickie

June 18, 2024

1 Introduction

TRAMPR is an R package for matching terminal restriction fragment length polymorphism (TRFLP) profiles between unknown samples and a database of knowns. TRAMPR facilitates analysis of many unknown profiles at once, and provides tools for working directly with electrophoresis output through to generating summaries suitable for community analyses with R's rich set of statistical functions. TRAMPR also resolves the issues of multiple TRFLP profiles within a species, and shared TRFLP profiles across species.

This document illustrates a sample session using TRAMPR, covering loading data, manipulating a database of known profiles, running an analysis, and extracting and manipulating results for further community analysis. Both TRAMPR and this document assume some knowledge of R. In particular, you should be familiar with "An Introduction to R" (included with your R distribution).

To get started, start R, then load the TRAMPR package:

```
> library(TRAMPR)
```

2 Loading data

The TRAMPR package includes functions to help convert data from Applied Biosystems Gene Mapper (ABI) output format into objects suitable for analysis ("TRAMPsamples" objects). The procedure is documented in the help file ?load.abi, and a worked example follows. Alternatively, data can be read from other formats if you reformat it manually; the help page ?TRAMPsamples explains the required format. We welcome contributions for routines for importing data from other formats.

We have included a demonstration data set with TRAMPR. To use this, copy all the files begining with "demo_samples_abi" from the directory where TRAMPR is installed into your current working directory. The required files are:

- demo_samples_abi.txt
- demo_samples_abi_template_full.csv
- demo_samples_abi_info_full.csv
- demo_samples_abi_soilcore.csv

These can be copied from within R, by typing:

```
> files <- c("demo_samples_abi.txt",
+            "demo_samples_abi_template_full.csv",
+            "demo_samples_abi_info_full.csv",
+            "demo_samples_abi_soilcore.csv")
> file.copy(system.file(files, package="TRAMPR"), ".")
```

Next, create the "template" file. Template files are required to record which enzymes were used for each run, since that is not included in the ABI output, and to group together separate runs (typically different enzymes) that apply to the same individual. The function load.abi.create.template will create a template that contains all the unique file names found in the ABI file (as sample.file.name), and blank columns titled enzyme and sample.index.

```
> load.abi.create.template("demo_samples_abi.txt")
```

```
Saved template file in /tmp/RtmpyDL1fD/Rbuild2df3a3ff826/TRAMPR/vignettes/demo_samples_abi_template.
```

The `enzyme` and `sample.index` columns need filling in, which can be done in Excel, or another spreadsheet program. The `sample.index` column links `sample.file.name` back to an individual sample; multiple `sample.file.name`s that share `sample.index` values come from the same individual sample. It is best to use positive integers here, but if you enter strings (e.g. `a1`, `b1`) these will be converted to integers for you (see the help file `?load.abi`). The file `demo_samples_abi_template_full.csv` contains a filled in version of the template file, so we will use that.

```
> file.rename("demo_samples_abi_template_full.csv",
+             "demo_samples_abi_template.csv")
```

Then, create an "info" file. This is optional, but useful if you want to associate extra information against your samples. The function `load.abi.create.info` will create an info file that contains all the unique values of `sample.index`, and an empty column titled `species`. The `species` column can be filled in where the species is known (e.g. from collections of sporocarps). Any additional columns may be added.

```
> load.abi.create.info("demo_samples_abi.txt")
```

```
Saved info file in /tmp/RtmpyDL1fD/Rbuild2df3a3ff826/TRAMPR/vignettes/demo_samples_abi_info.csv
```

The file `demo_samples_abi_info_full.csv` contains a filled in version of the info file, so we will use that.

```
> file.rename("demo_samples_abi_info_full.csv",
+             "demo_samples_abi_info.csv")
```

We added a column `soilcore.fk`, which indicates which soil core the samples came from. An additional csv file `demo_samples_abi_soilcore.csv` contains information about the soil cores (forest type, location, etc.). We read that data in so that this can be added into the `TRAMPsamples` object:

```
> soilcore <- read.csv("demo_samples_abi_soilcore.csv", stringsAsFactors = TRUE)
```

The last thing that needs specifying is how to interpret the `dye` column in the ABI output. We assume that dyes can be consistently translated to primers (i.e. a particular dye always corresponds to the same primer). The list below indicates that the dye `"B"` refers to the primer "ITS1F", while the dyes `"G"` and `"Y"` both refer to the primer "ITS4".

```
> primer.translate <- list(ITS1F="B", ITS4=c("G", "Y"))
```

Now, use `load.abi` to construct a `TRAMPsamples` object containing all the samples. Because we pass `soilcore` as a named argument, the resulting `TRAMPsamples` object will have a `soilcore` element. Note that we are passing the `soilcore` *object* (which is a data frame) rather than the filename.

```
> demo.samples <- load.abi("demo_samples_abi.txt",
+                          primer.translate=primer.translate,
+                          soilcore=soilcore)
```

```
Found info file at demo_samples_abi_info.csv
```

2.1 Knowns

`TRAMPR` comes with a set of demonstration "knowns", which we will use here. Saving and managing knowns database is covered later in this document (Section 5). To load the demonstration knowns, use:

```
> data(demo.knowns)
```

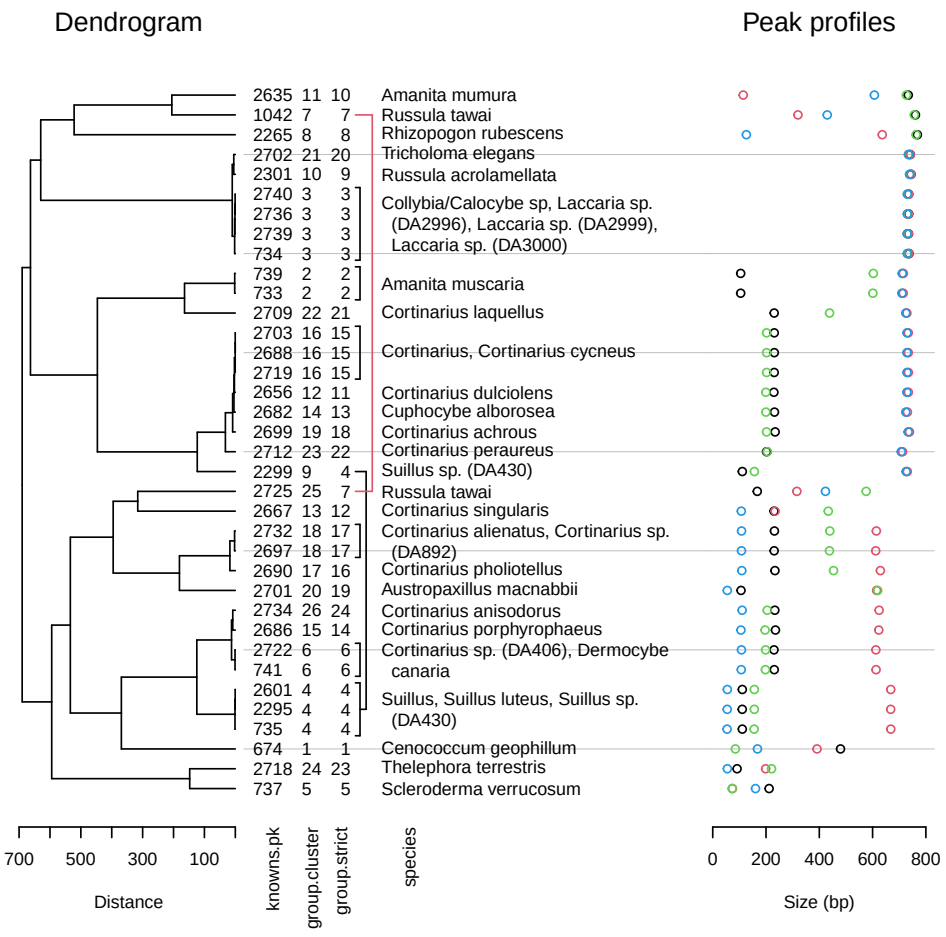

Figure 1: A plot summarising the knowns database `demo.knowns`. The panels (from left to right) illustrate: (1) the clutering of knowns based on TRFLP profile similarity, (2) known names, and group indices, and (3) the TRFLP profile for each individual (different colours indicate different enzyme/primer combinations).

`TRAMPR` allows species in the knowns database to have multiple profiles. The TRFLP profile of a single species can have variation in peak sizes due to DNA sequence variation; profiles for different individuals may be slightly or completely different. Conversely, two species may have indistinguishable TRFLP profiles. If knows are not grouped to take account of this, the number of knowns per sample is likely to be overestimated. `TRAMPR` groups knowns where their TRFLP profiles are very similar (based on clustering), or where knowns in the database share the same name. Plotting a knowns database illustrates how grouping occurs (see Figure 1 for output).

```
> plot(demo.knowns)
```

3 Find matches with `TRAMP`

Depending on the size of the samples and knowns databases, and the speed of your computer, running `TRAMP` may be slow. Optional arguments to `TRAMP` control the strictness of the match (e.g. number of base pairs difference between sample and knowns profiles); see the help file `?TRAMP` for more information. To run `TRAMP`, run:

```
> fit <- TRAMP(demo.samples, demo.knowns)
```

A fit may be plotted to inspect the match. For example, to see the fit for sample 565, do (See Figure 2 for output):

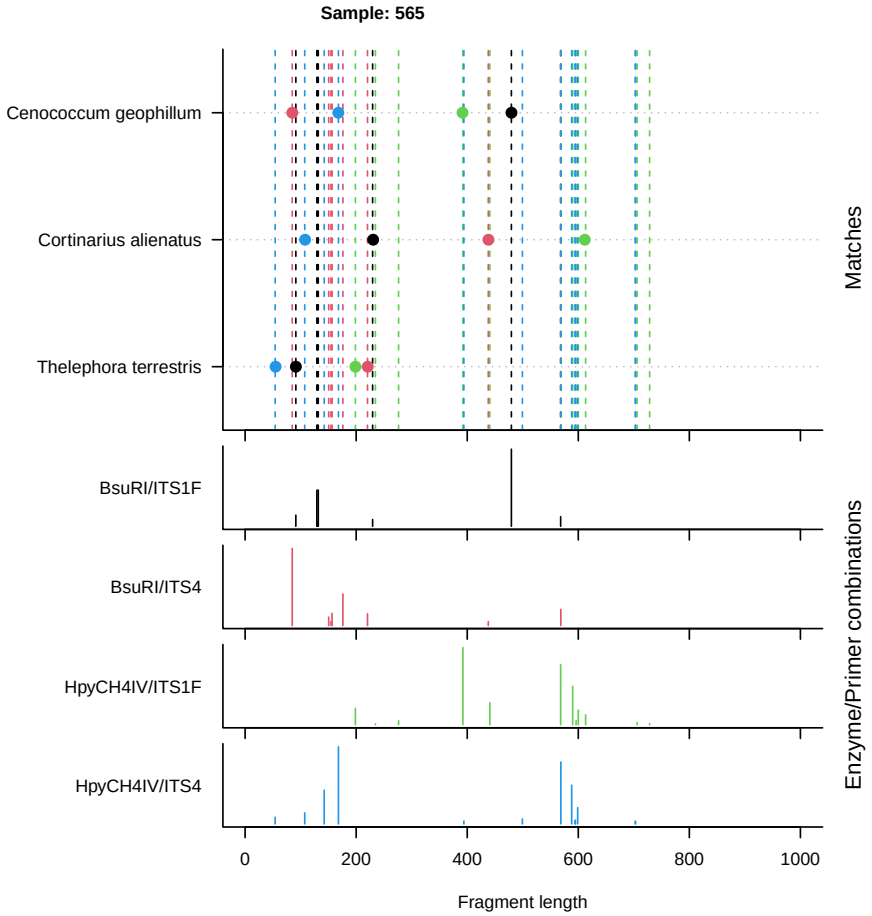

Figure 2: A plot showing the fit of sample 565 against a database of knowns. The sample matches nine knowns, and many unmatched peaks remain. The square points indicate matches between the knowns (listed on the y axis of the top panel) and the peaks of the sample. The peak profile across four enzyme/primer combinations is displayed in the bottom four panels.

```
> plot(fit, 565)
```

You can export a PDF containing one fit per page:

```
> pdf("TRAMP_fits.pdf")
> plot(fit)
> dev.off()
```

The main output of a `TRAMP` analysis is a presence/absence matrix, produced by `summary`:

```
> m <- summary(fit)
> m[1:5, 1:5]
```

```
      known
sample   674   733   734   735   737
   101 FALSE FALSE FALSE FALSE FALSE
   102 FALSE FALSE FALSE FALSE FALSE
   110  TRUE FALSE FALSE FALSE FALSE
   111 FALSE FALSE FALSE FALSE FALSE
   117 FALSE FALSE  TRUE FALSE FALSE
```

`TRAMPR` can also use "unknown knowns"; these are patterns that repeatedly occur in TRFLP data, but for which a species is currently unidentified. The function `build.knowns` attempts to spot good "unknown knowns" in your data set:

```
> knowns.unk <- build.knowns(demo.samples)
```

`build.knowns` is also useful where your sample data include profiles from sporocarps. You can use the argument `restrict=TRUE`, and relax other conditions to generate a knowns database from only your sporocarps (see the help page `?build.knowns`) for more information. (The command below will not run on the demonstration data provided, as none of the samples come from sporocarps.)

```
> knowns.sporocarps <- build.knowns(demo.samples, restrict=TRUE,
+                                   min.ratio=0)
```

The new knowns database can be added to either the `TRAMP` object or the `TRAMPknowns` object. To add the new knowns to the `TRAMP` object:

```
> fit <- combine(fit, knowns.unk)
```

Alternatively, to add the new knowns to the `TRAMPknowns` object:

```
> demo.knowns <- combine(demo.knowns, knowns.unk)
```

These are equivalent in their actions on the knowns. The `knowns` element of `fit` and `demo.knowns` will be the same:

```
> identical(demo.knowns, fit$knowns)
```

```
[1] TRUE
```

The main difference is that if the knowns are added to the `TRAMP` object, then the fit will be rerun automatically. You can always extract the modified knowns from a `TRAMP` object:

```
> demo.knowns <- fit$knowns
```

`build.knowns` is fairly conservative and may miss some potential "unknown knowns". In particular, "split peaks", where the ABI program identifes two large "peaks" occuring within a few base pairs of each other where only one true peak exists, will often be missed. You can plot the samples that were not added to the knowns database by `build.knowns` like this:

```
> samples <- setdiff(labels(demo.samples), labels(demo.knowns))
> plot(fit, samples)
```

Sample 221 shows a clear split peak for BsuRI/ITS1F, and may be a good potential known:

```
> plot(fit, 221)
```

or

```
> plot(demo.samples, 221)
```

will show you the profile (not shown here). We can add this to the database using the `add.known` function. As with `combine`, this can be used directly on the `TRAMP` object or on a knowns database. We will use the `TRAMP` object again (this generates a warning, since there are multiple peaks for BsuRI/ITS1F).

```
> fit <- add.known(fit, 221, prompt=FALSE)
```

Using the `summary` function, we can see which samples match this new known:

```
> names(which(summary(fit)[,"221"]))
```

```
 [1] "221" "224" "310" "311" "339" "579" "580" "583" "585" "586"
```

Plotting the last of these shows the new known plus two *Amanita muscaria* knowns (and several other "unknown knowns" from running `build.knowns`) matching the sample (see Figure 3).

```
> plot(fit, 586)
```

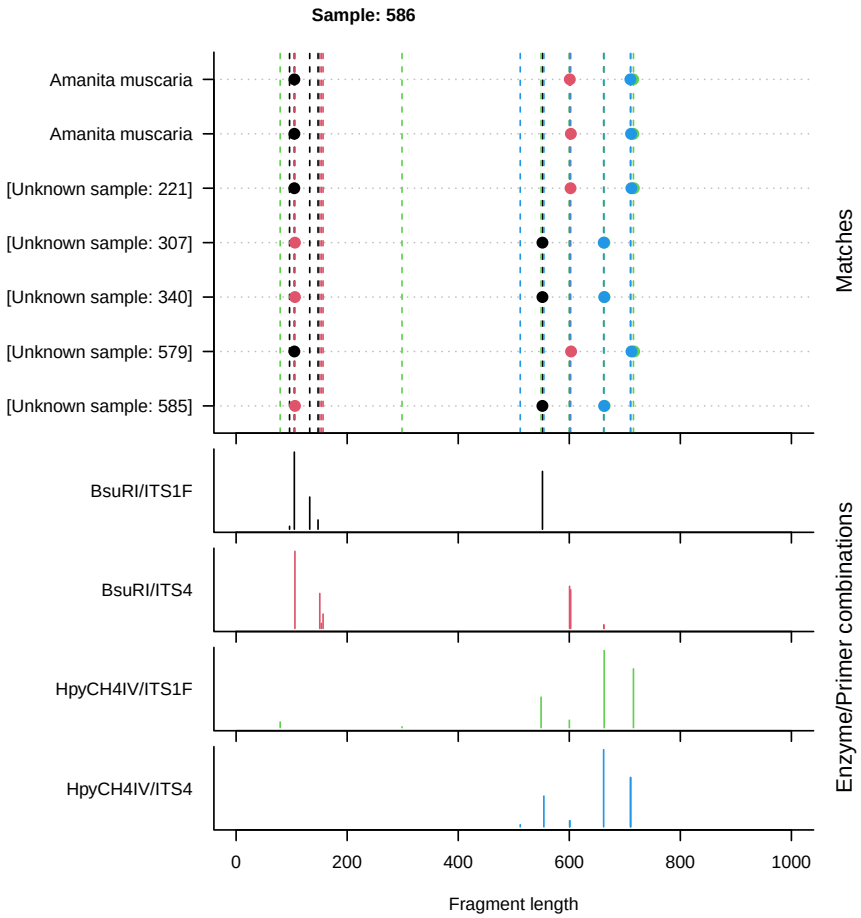

Figure 3: A plot showing the fit of sample "586" against a database of knowns, showing the sample matching *Amanita muscaria*, plus several "unknown knowns", including one added manually ("Unknown sample: 221"). In this case, the Unknown sample 221 appears to be a new *Amanita muscaria*, and this is confirmed by grouping (see Figure 4).

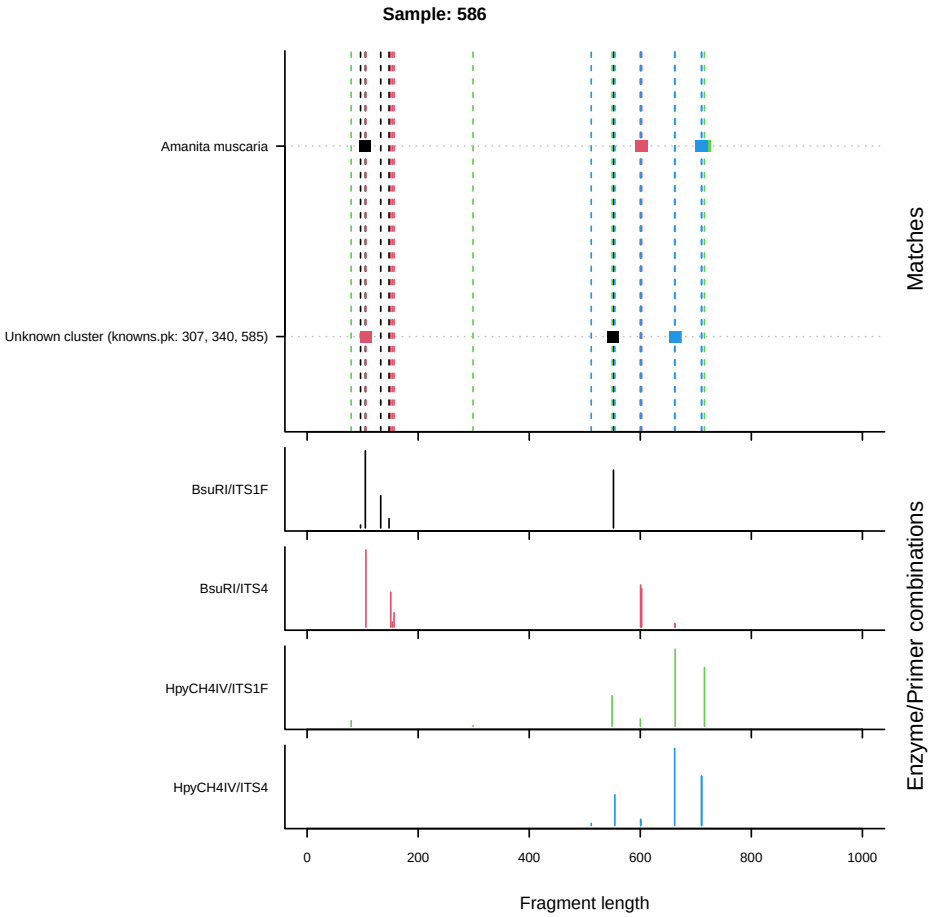

Figure 4: A plot showing the fit of sample "586" against a database of knowns, but with the knowns grouped (compare with Figure 3). The unknown known has grouped with *Amanita muscaria*. Another group of "unknown knowns" also matches this sample.

This plot illustrates the need for grouping; since there are multiple entries for *Amanita muscaria*, it will be counted as a match twice. Using `grouped=TRUE`, we can group TRFLP profiles that are very similar or that share species names:

```
> plot(fit, 586, grouped=TRUE)
```

Output is shown in Figure 4; with grouping, the matches collapse into two separate groups. Unknown sample 221 groups within the *Amanita muscaria* group. By allowing multiple entries for single species, we can keep the error size used for matching low (there is a discussion of this in the help file `?group.knowns`).

The `update.TRAMP` function can also interactively add knowns. See the help file (`?update.TRAMP`), or just try (output not shown):

```
> fit <- update(fit)
```

4 Using TRAMP results

Here we demonstrate a simple principle components analysis of the community composition of the sample dataset. [This is not neccesarily the appropriate analysis type for this kind of data, and we do not endorse it.] Once the presence/absence matrix is extracted from the `TRAMP` object, this uses just built-in R functions. First, we construct a presence/absence matrix, and restrict this to knowns where at least one sample matches. We are using the grouped presence/absence matrix to include only genuinely different species.

```
> m <- summary(fit, grouped=TRUE)
> m <- m[, colSums(m) > 0]
```

Next, aggregate the results by soilcore. Several samples came from each soil core, and a known is scored as being present if it is present in `any` of the samples. This produces a much smaller matrix (see output in Figure 5).

```
> cores <- fit$samples$info$soilcore.fk
> m.bycore <- aggregate(m, by=list(cores=cores), any)
> rownames(m.bycore) <- m.bycore$cores
> m.bycore <- m.bycore[-1]
> m.bycore
```

(The `m.bycore[-1]` line removes an index column that `aggregate` creates).

Now, determine which vegetation type applies to each row in the presence/absence matrix.

```
> core.veg <- soilcore$vegetation[match(rownames(m.bycore),
+                                       soilcore$soilcore.pk)]
```

Next, create a rank-abundance diagram of the data (see Figure 6 for output).

```
> sp.freq <- sort(colSums(m.bycore), decreasing=TRUE)
> plot(sp.freq, xlab="Species rank", ylab="Species frequency", type="o",
+      pch=19)
```

You can also calculate mean species richness across the different vegetation types:

```
> tapply(rowSums(m.bycore), core.veg, mean)
```

```
Nothofagus solandri      Pinus contorta
                3.0                 1.5
```

Finally, run a principle components analysis, and create a plot of the first two principle axes. To distinguish between the two different forest types, we determine the vegetation type for each core, and create a vector of colours so that cores from *Nothofagus solandri* stands are blue circles and cores from *Pinus contorta* stands are red crosses (see Figure 7 for output).

```
> pca <- prcomp(m.bycore)
> col <- c("blue", "red")[as.integer(core.veg)]
> pch <- c(1, 3)[as.integer(core.veg)]
> plot(pca$x[,1], pca$x[,2], col=col, pch=pch, xlab="PC1", ylab="PC2")
> legend("top", levels(core.veg), col=c("blue", "red"), pch=c(1, 3))
```

```
       1     2     3     4     5     6     7     8     9    10    11    19
2  FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
3  FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
4  FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE
5   TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE
6   TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE
12 FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
13 FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
14 FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
15 FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
16 FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
17 FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE
18  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
19 FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
20 FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
21 FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
32 FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
33 FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
34 FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
35 FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
36 FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
      20    22    23    24    28    29
2  FALSE FALSE  TRUE FALSE FALSE  TRUE
3  FALSE FALSE FALSE  TRUE FALSE FALSE
4  FALSE FALSE FALSE FALSE FALSE FALSE
5   TRUE FALSE FALSE FALSE FALSE FALSE
6  FALSE  TRUE FALSE FALSE FALSE FALSE
12 FALSE FALSE FALSE FALSE FALSE FALSE
13 FALSE FALSE FALSE FALSE FALSE FALSE
14 FALSE FALSE FALSE FALSE FALSE FALSE
15 FALSE FALSE FALSE FALSE FALSE FALSE
16 FALSE FALSE FALSE FALSE FALSE FALSE
17 FALSE  TRUE FALSE FALSE FALSE FALSE
18 FALSE  TRUE FALSE FALSE FALSE FALSE
19 FALSE FALSE FALSE FALSE FALSE FALSE
20 FALSE  TRUE FALSE FALSE  TRUE FALSE
21 FALSE FALSE FALSE FALSE FALSE FALSE
32 FALSE FALSE FALSE FALSE FALSE FALSE
33 FALSE FALSE FALSE FALSE FALSE FALSE
34 FALSE FALSE FALSE FALSE FALSE FALSE
35 FALSE FALSE FALSE FALSE FALSE FALSE
36 FALSE FALSE FALSE FALSE FALSE FALSE
```

Figure 5: Presence/absence matrix for the soil core data. Rows represent different soil cores, with row names corresponding to `soilcore.pk` values in the `soilcore` table. Columns correspond to groups of knowns, each of which may represent multiple knowns entries.

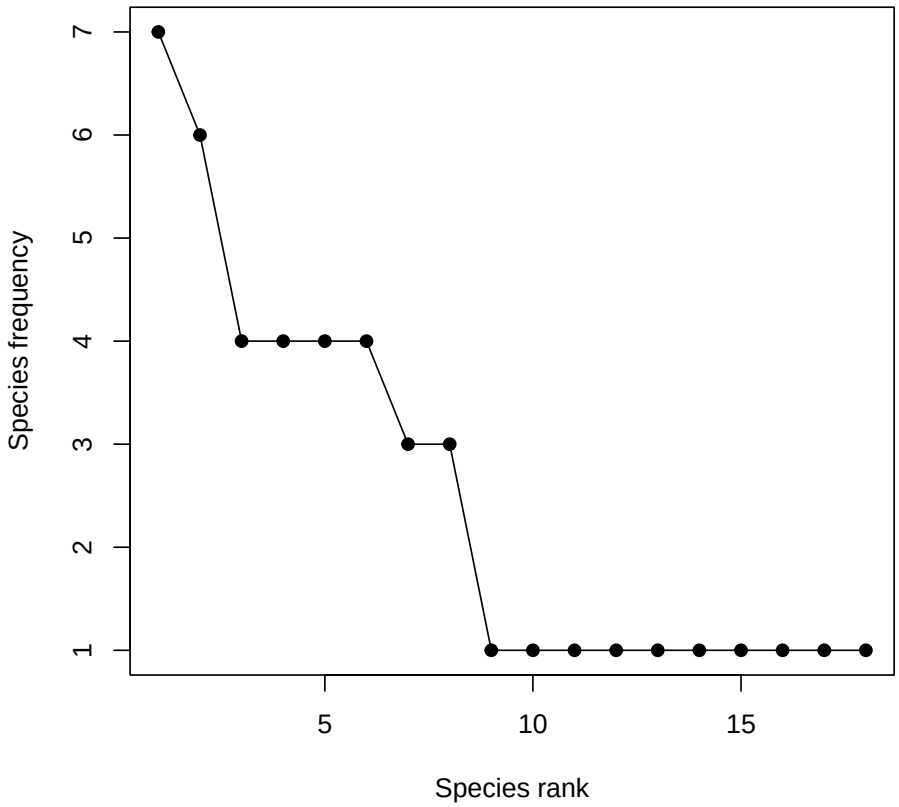

Figure 6: Rank-abundance diagram for 18 species present across 20 soilcores.

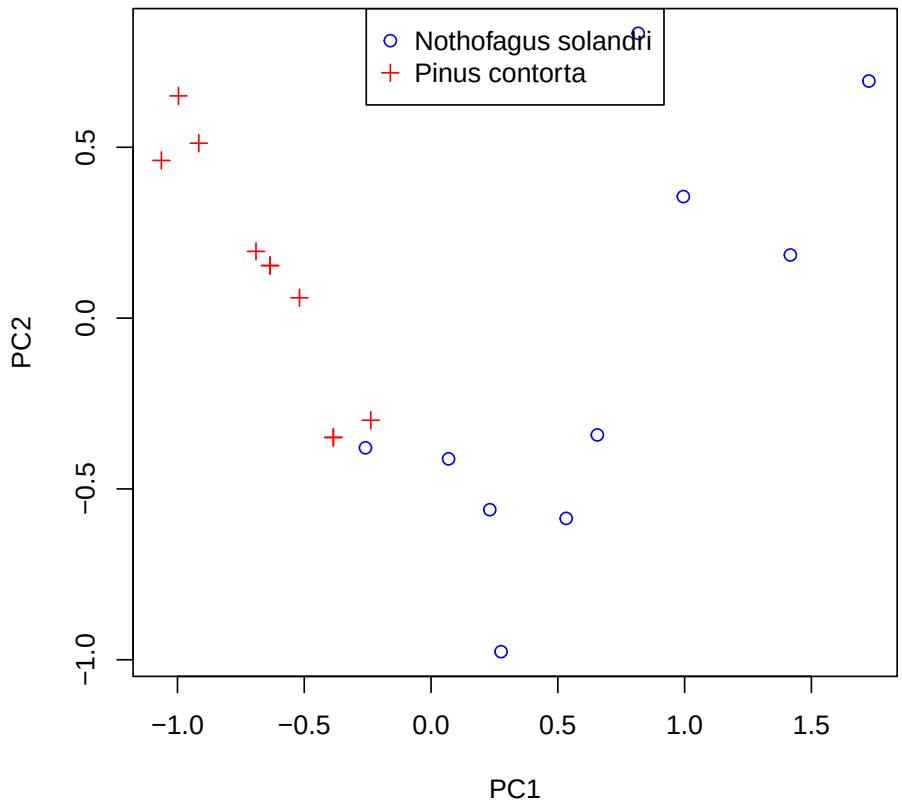

Figure 7: Principle components analysis of soil community data. Different points represent different soil cores, and colours distinguish forest types. [We are not endorsing PCA as an appropriate analysis for these data.]

5 Managing and saving knowns

We can save the modified knowns database in two ways. If you want to edit any aspect of the knowns (e.g. species names), or update an external database, use `write.TRAMPknowns`:

```
> write.TRAMPknowns(fit$knowns, "my_knowns")
```

This will create files `my_knowns_info.csv` and `my_knowns_data.csv`, which contain the information required to build the knowns. The files are documented in `?write.TRAMPknowns`, and can be edited in Excel[1]. The knowns can be reloaded in a future R session by:

```
> my.knowns <- read.TRAMPknowns("my_knowns")
```

Alternatively, knowns can be saved like any other R object, using `save` and `load`. To save your knowns, do:

```
> my.knowns <- fit$knowns
> save(my.knowns, file="my_knowns.Rdata")
```

This will create a single file `my_knowns.Rdata`, which is a non-human readable Rdata file containing the knowns object. Note that the object saved must be an object named in the global environment, so we must extract the knowns first.

To reload this in a future R session, do:

```
> load("my_knowns.Rdata")
```

This will create an object called `my.knowns` in the global environment.

6 Concluding remarks

This is a brief introduction, but which should cover most of the functionality of the `TRAMPR` package. Detailed information can be found in the reference manual; type `library(help=TRAMPR)` for a list of topics. In particular, the behaviour of most functions can be altered by changing optional arguments.

[1] If editing in Excel, when saving it will warn you about losing information when saving as a csv file – ignore these warnings, as `TRAMP` requires that the data are saved as csv.