The Journal of Open Source Software

Archeofrag: an R package for refitting and spatial analysis in archaeology. Presentation and tutorial

Sébastien Plutniak[1]

1 TRACES Laboratory, Toulouse, France

DOI: 10.21105/joss.04178

Software

- Review
- Repository
- Archive

Editor: Pending Editor

Submitted: 17 February 2022
Published: 17 February 2022

Summary

Distinguishing between spatial entities is fundamental in archaeology since archaeologists deal with spatial phenomena at multiple scales of analysis. During an excavation, objects are discovered within various types of spatial units e.g., stratigraphic layers, pits, hearths and houses. Spatial units are far from being raw data, and the identification and determination of their boundaries is the result of conjoint lines and methods of investigation, to name only a few: field observations, geoarchaeology, sedimentology, and the study of archaeological "refits." Refitting fragments belonged to the same object at some moment in the past. More precisely, archaeologists deduce former connection relationships from the symmetry and the possibility of contact of significantly large surface areas from two fragments, which can be physically adjusted (the fragments "refit"). Here, "connection" is used as a shorthand to refer to the connection relationship that existed in the past between two areas of an object before they were broken into fragments. Archaeological refitting analysis has several aims: 1) to reconstruct objects, 2) to determine technical sequences (e.g., stone tool manufacture), and 3) to determine the reliability of spatial units and their possible admixture due to pre- and post-depositional processes. This analysis has long been used for the latter aim, benefiting from multiple methodological improvements (for an overview, see Cziesla et al. (1990), Schurmans & De Bie (2007)). These methods have relied on the comparison between the number of refits between different spatial units and within these units. However, it has been demonstrated that considering the number of refits without considering their topology can lead to misleading interpretations. A method, coined TSAR "Topological Study of Archaeological Refitting," was developed to overcome this issue using graph theory to model the topology of the relations between fragments (Plutniak (2021b), Plutniak (2022)). This renewed approach distinguishes between ambiguous cases (Figure 1), and is much more robust and less sensitive to the lack of information than count-based methods, thus resulting in a more accurate evaluation of the reliability of the boundaries between spatial units. `Archeofrag` is an R package (R Core Team, 2020) implementing the TSAR method.

The Journal of Open Source Software

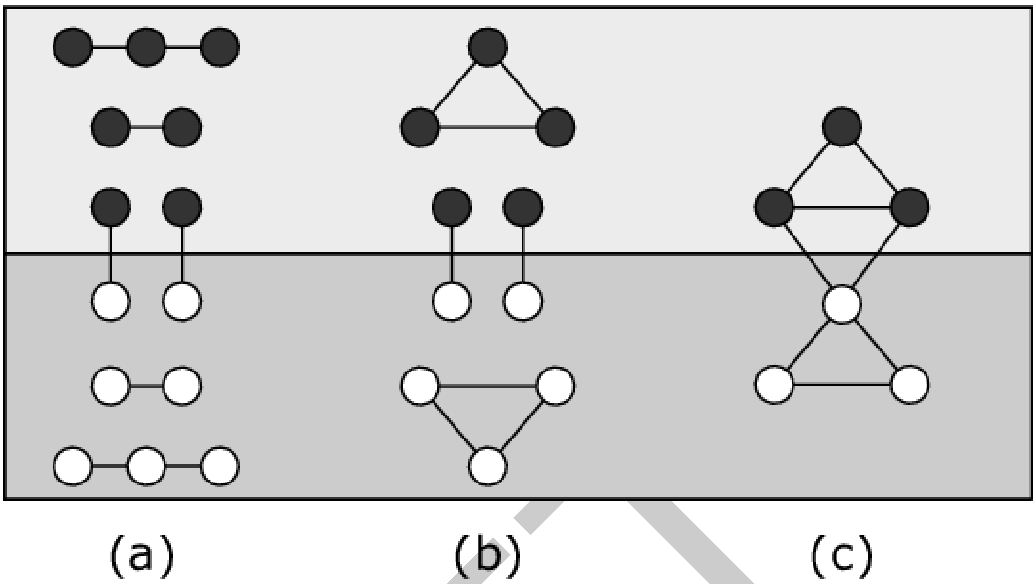

Figure 1: Three examples (a-c) of two layers with internal refitting (n=6) and inter-layer refitting (n=2). Although the numbers of relationships are equal in all examples, their archaeological interpretation are very different: relevant distinction between the two layers in (a); relevant distinction with higher confidence about the fragmented objects' initial location in (b); doubtful distinction between layers in (c).

Statement of need

The use of R in archaeology has increased slowly albeit constantly, during the last two decades. However, the development of R packages for specific archaeological needs is an even more recent phenomenon. Only a few packages are available for spatial analysis in archaeology, notably for stratigraphic analysis:

- `stratigraphr`: package in its early development phase to visualise and analyse stratigraphies as directed graphs (Harris matrices).
- `tabula`: generic package to visualise remain counts, which can also be used to compare layers (Frerebeau, 2019).
- `recexcavAAR`: package for 3D reconstruction and analysis of excavations (Schmid & Serbe, 2017).
- `archaeoPhases`: package for Bayesian analysis of archaeological chronology to define stratigraphic phases (Philippe & Vibet, 2020).

`Archeofrag` complements this series of packages with a specific focus on refitting.

Package overview

`Archeofrag` mainly uses the `igraph` library for graph analysis (Csárdi & Nepusz, 2006) and also relies on some functions from the `RBGL` package (Carey et al., 2020). It comes with an example data set (Plutniak, 2021a) containing refitting data on the pottery found during excavations at Liang Abu rock shelter, in Borneo (Plutniak et al., 2016).

`Archeofrag` has six main sets of functions:

1. **Data management**: create, check, and transform fragmentation graphs (including edge weighting based on the topological properties of the vertices and optionally, on the size of the fragments and the distance between the locations where they were discovered during excavation)
2. **Visualisation**: represent fragmentation graphs as node-and-edge diagrams
3. **Boundary-related statistics**: count the relationships within and between two spatial units, measure their cohesion and the admixture values
4. **Spatial unit-related statistics**: characterise the topology of a specific set of refitting relationships (e.g., a layer) with several measurements
5. **Simulation**: generate simulated fragmentation graphs, simulate their alteration (missing data); compare an empirical graph with similar simulated graphs and return results in a convenient way
6. **Similarity analysis**: in addition to the topological analysis of refits, functions are available to analyse similarity relationships, which are determined between fragments considered as sharing enough common features (motif, clay, inclusions, etc.) to state they are (and were) parts of the same initial object.

Data management

`Archeofrag` is intended to be used with two sources of data, namely the user's empirical data and artificially generated data using its simulation function. User's data must be split into different tables:

- **Fragment table**: each line contains the unique identifier of a fragment, its spatial unit, and optionally additional information.
- **Connection table**: an edge list with the identifiers of two connected fragments by line.
- **Similarity table** (optionally): each line includes the unique identifier of a fragment and the identifier of a set of similar fragments it belongs to.

The package includes functions to generate summary statistics about the fragmentation graph and extract specific sub-graphs (by layer, by component size, etc.).

Visualisation of fragmentation graphs

The fragmentation graphs are visualised as node-and-edge diagrams. For graphs with only two spatial units and connection relationships, the location of the nodes in the upper and the lower part of the plot are based on their spatial unit.

Boundary-related statistics: measuring the cohesion and admixture of two spatial units

Evaluating the consistency of spatial units and their division from refits is the first aim of the `Archeofrag` package. Evaluation follows a three-step procedure, implemented in three related functions:

1. Weighting the edges. Three parameters can be combined: the topology of the connection relationships (mandatory), the size of the two connected fragments (optional), and the spatial distance between the location where they were found (optional). The optional parameters implement the hypothesis that two large fragments found far from each other suggest more disturbance in the site than two small fragments found very close together (see an application and details in Caro et al., 2022).

2. Measuring the internal cohesion of each spatial unit (intuitively, how they are "self-adherent" to themselves)
3. Measuring the admixture of two spatial units (a summary statistic based on two cohesion values)

Cohesion and admixture values are computed by pairs of spatial units. Results for cohesion measurements range between [0;1], with values towards 0 for low cohesion and towards 1 for high cohesion, with their sum never being superior to 1 for a pair of layers. Results for admixture measurements range between [0;1].

Spatial unit-related statistics: fragmentation patterns, technology, human behaviour

The second aim of the TSAR method implemented in `Archeofrag` is to characterise spatial units based on the topological properties of the connection relationships between the fragments they contain. Several functions are provided for this purpose, selected for their relevance in the archaeological context, namely cycle count, path length, and component diameter (a component is related to an initial object).

The archaeological interpretation of the numerical values depends on the type of object (lithic, pottery, etc.) and their completeness or incompleteness. These values can suggest specific behaviours related to the production or use of the objects (intentional breaking), and post-depositional processes (natural breaking, scattering). This aspect of the TSAR method will be further developed in the future.

Simulation of fragmentation graphs

The simulation function generates connected fragments scattered within one or two spatial units (see Plutniak (2021b) for details). It can be set with multiple parameters (number of initial objects/fragments, number of fragments, number of relationships, number of fragments and relationships, the balance between layers, etc.). It can also be constrained to generate only planar graphs since this corresponds to the fragmentation in some specific archaeological contexts (e.g., pottery with simple shapes, small sets of refits). However, the run time of this function is doubled when this constraint is used.

Use case: pottery from Liang Abu rock shelter

Building the fragmentation graph

The `Archeofrag` package comes with a small example data set called "Liang Abu," related to the pottery fragments found on the surface and in the first two layers of the Liang Abu rock shelter (Plutniak, 2021a). The data set contains three data frames:

- a table with information about the fragments (a unique identifier, the layer, their length and width, etc.),
- a table with the connection relationships between these fragments (each row contains the unique identifiers of two refitting fragments),
- a table with the similarity relationships between these fragments (two fragments are termed "similar" if they seem to come from the same object but do not have connecting edges.

The Journal of Open Source Software

The `make_frag_object` function builds objects with the class "frag." Frag objects are not required by the other `Archeofrag` functions, however, using them ensures that the data are suitable for the next steps of the analysis. The `make_cr_graph` function takes a frag object and generates an `igraph` graph object representing the connection relationships.

```
library(archeofrag)
data(LiangAbu)
abu.frag <- make_frag_object(cr=df.cr, fragments=fragments.info)
abu.g <- make_cr_graph(abu.frag)
```

Visualisation and subgraph extraction

Several `Archeofrag` functions ensure that the first examination of the data is easy. The `frag.relations.by.layers` function returns a matrix with the number of relationships within and between spatial units (e.g., stratigraphic layer).

```
frag.relations.by.layers(abu.g, "layer")
```

```
##
##       0  1  2
##    0  4
##    1  0 18
##    2  0  3 31
```

The diagonal of the matrix gives the number of intra-layer relationships, and the other values refer to inter-layer relationships. Here, for example, there are 31 connection relationships within layer 2, and 3 connection relationships between layers 1 and 2. No connection relationship was found between the surface ("0") and layer 2.

The `frag.graph.plot` function generates a visual representation of the graph:

```
frag.graph.plot(abu.g, layer.attr="layer", main="All layers")
```

The fragments are coloured by layer and the three inter-layer relationships can be observed.

Let us now focus on layers 1 and 2. The `frag.get.layers.pair` function allows the user to extract a pair of layers.

```
abu.g12 <- frag.get.layers.pair(abu.g, layer.attr="layer",
                                sel.layers=c("1", "2"))
```

This subgraph is drawn with the `frag.graph.plot` function:

```
frag.graph.plot(abu.g12, layer.attr="layer", main="Layers 1 and 2")
```

The function has a different behaviour if applied to a fragmentation graph with only two spatial units: the nodes are vertically localised to reflect their location in the two spatial units. In addition, note that standard `plot` arguments can be passed to the `frag.graph.plot` function, e.g., the `main` argument to define the plot's title.

The `frag.get.layers.pair` function has additional parameters to set the minimum size of the connected fragments sets (`size.mini`) and to extract only the sets of connected fragments which include relationships between the two spatial units (`mixed.components.only`).

```r
frag.get.layers.pair(abu.g, layer.attr="layer", sel.layers=c("1", "2"),
                     size.mini=2, mixed.components.only=TRUE)
```

```
## IGRAPH d4b5063 UN-- 19 22 --
## + attr: frag_type (g/c), name (v/c), layer (v/c), zmin (v/n), zmax
## | (v/n), square (v/c), sherd.type (v/c), thickness (v/n), length (v/n),
## | membership (v/n), type_relation (e/c)
## + edges from d4b5063 (vertex names):
##  [1] 187--188 165--195 195--196 195--197 196--198 195--204 196--204 197--204
##  [9] 198--204 195--25  188--250 27 --28  27 --366 27 --367 28 --367 366--367
## [17] 27 --371 332--371 366--371 25 --8   28 --835 835--836
```

Additionally, the `frag.get.layers` function can extract a set of specified spatial unit(s), e.g., the refits within the first layer at Liang Abu:

```r
frag.get.layers(abu.g, layer.attr="layer", sel.layers="1")
```

```
## $`1`
## IGRAPH 8005ac2 UN-- 23 18 --
## + attr: frag_type (g/c), name (v/c), layer (v/c), zmin (v/n), zmax
## | (v/n), square (v/c), sherd.type (v/c), thickness (v/n), length (v/n),
## | type_relation (e/c)
## + edges from 8005ac2 (vertex names):
##  [1] 123--124  187--188  195--196  195--197  196--198  195--204  196--204
##  [8] 197--204  198--204  195--25   301--302  313--314  392--408  435--441
## [15] 477--478  25 --8    435--9999 441--9999
```

Edge weighting, cohesion and admixture computation

Weighting the edges is a crucial step in the TSAR / `Archeofrag` approach because it integrates the topological properties of the fragmentation graph. The `frag.edges.weighting` function assigns a value to each edge based on the topological properties of the vertices this edge connects.

```r
abu.g12 <- frag.edges.weighting(abu.g12, layer.attr="layer")
```

Then, the `frag.layers.cohesion` function is used to calculate the cohesion value of each layer.

```r
frag.layers.cohesion(abu.g12, layer.attr="layer")
```

```
##     cohesion1 cohesion2
## 1/2 0.3977727 0.5927749
```

These values determine the cohesion (self-adherence) of the spatial units (here, layers) based on the distribution of the refitting relationships. Note that the weighting of the edges is mandatory for the computation of cohesion. Using the `frag.layers.cohesion` function on a non-weighted fragmentation graph will give an error.

In addition to topological properties, the computation of edge weights can optionally include other parameters, namely the morphometry of the fragments and the distance between the location where they were found. In the following example, the length of the pottery sherds is used as a morphometric proxy:

```r
abu.g12morpho <- frag.edges.weighting(abu.g12,
                                      layer.attr="layer",
                                      morphometry="length")
```

Using the morphometry parameter results, layer 2 is more cohesive than layer 1:

```r
frag.layers.cohesion(abu.g12morpho, layer.attr="layer")
```

```
##     cohesion1 cohesion2
## 1/2 0.3263172  0.666898
```

In addition, the `frag.layers.admixture` function returns a value quantifying the admixture between the two layers. Let us compare the results obtained when the morphometry is used or not:

```r
# topology-based weighting:
frag.layers.admixture(abu.g12, layer.attr="layer")
```

```
##   admixture
## 0.009452435
```

```r
# topology + morphometry weighting:
frag.layers.admixture(abu.g12morpho, layer.attr="layer")
```

```
##   admixture
## 0.006784769
```

In this case, using the morphometry in the computation lowers the admixture between layers 1 and 2 at Liang Abu.

Testing layer formation hypotheses using simulated data

Simulation-based hypotheses can be tested by combining the functions offered by `Archeofrag`.

Generating artificial fragmentation graphs

The `frag.simul.process` function generates a pair of spatial units containing fragmented objects with connection relationships within and between these units. The next command creates two spatial units populated with 20 initial objects (corresponding to the "connected components" of a graph) which are fragmented into 50 pieces.

```r
simul.g <- frag.simul.process(n.components=20, vertices=50)
```

This illustrates the simplest use of the `frag.simul.process` function, which has several other parameters to control the features of the simulation.

The number of initial spatial units is a crucial parameter, set using the `initial.layers` parameter with "1" or "2." This parameter determines the method used to construct the graph and, accordingly, the underlying formation process hypothesis.

If `initial.layers` is "1," the fragmentation process is simulated assuming that all the objects were originally buried in a single spatial unit. The two clusters observed at the end of the process are due to fragmentation and displacement.

1. A single spatial unit is populated with the initial objects,
2. the fragmentation process is applied,
3. spatial units are assigned to the fragments,
4. some fragments are moved as determined by the value of the `disturbance` parameter.

If `initial.layers` is "2," it assumes that the objects were buried in two different spatial units, which were later partially mixed due to fragmentation and displacement:

1. two spatial units are populated with the initial objects (components),
2. the fragmentation process is applied,
3. disturbance is applied.

The `vertices` and `edges` parameters are related: at least one of them must be set, or both (only if `initial.layers` is set to 1). Note that using both parameters at the same time increases the constraints and reduces the number of possible solutions to generate the graph. When there is no solution, an error occurs and a message suggests how to change the parameters.

The `balance` argument determines the number of fragments in the smaller spatial unit (**before** the application of the disturbance process). The `components.balance` also determines the contents of the two spatial units by affecting the distribution of the initial objects (components). Note that this argument is used only when `initial.layers` is set to 2.

The `aggreg.factor` parameter affects the distribution of the sizes of the components: this distribution tends to be more unequal when `aggreg.factor` has values close to 1.

By default, fragments from two spatial units can be disturbed and moved to another other spatial unit. However, the `asymmetric.transport.from` can be used to move fragments from only one given spatial unit.

Finally, the `planar` argument determines if the generated graph has to be planar or not (a graph is planar when it can be drawn on a plane, without edges crossing).

An example of a complete configuration of the function is:

```
frag.simul.process(initial.layers=1,
                   n.components=20,
                   vertices=50,
                   edges=40,
                   balance=.4,
                   components.balance=.4,
                   disturbance=.1,
                   aggreg.factor=0,
                   planar=T,
                   asymmetric.transport.from="1")
```

An additional function is intended to simulate the failure of an observer to determine the relationships between fragments. The `frag.observer.failure` function takes a fragmentation graph and randomly removes a given proportion of edges.

```
frag.observer.failure(abu.g12, likelihood=0.2)
```

Testing hypotheses

The versatile `frag.simul.process` function can generate fragmentation graphs under multiple hypotheses about the initial conditions (number of initial objects, number of initial spatial units, etc.). Testing measurements on observed empirical data against measurements made under these hypotheses can determine the most likely initial conditions and fragmentation process.

Here, this is illustrated by comparing measurements from Liang Abu layers 1 and 2 with measurements from simulated data under two hypotheses about the number of initial spatial units (e.g., layers), using the `initial.layers` parameter with two values, namely one or two initial layers.

A fragmentation graph is generated for each `initial.layers` value, using the parameters observed in the Liang Abu layers 1 and 2 fragmentation graph. Setting the simulator is made easier by using the `frag.get.parameters` function, which takes a graph and computes a series of parameters that are returned as a list.

```
params <- frag.get.parameters(abu.g12, layer.attr="layer")

# for H2:
test.2layers.g <- frag.simul.process(initial.layers=2,
                                     n.components=params$n.components,
                                     vertices=params$vertices,
                                     disturbance=params$disturbance,
                                     aggreg.factor=params$aggreg.factor,
                                     planar=params$planar)
# for H1:
test.1layer.g <- frag.simul.process(initial.layers=1,
                                    n.components=params$n.components,
                                    vertices=params$vertices,
                                    disturbance=params$disturbance,
                                    aggreg.factor=params$aggreg.factor,
                                    planar=params$planar)
```

Let us now generate not only one graph, but a large number of graphs to statistically compare measurements in the empirical and simulated graphs. The `frag.simul.process` function is set for the "two initial layers" hypothesis and embedded into an *ad hoc* function:

```
run.test2 <- function(x){
  frag.simul.process(initial.layers=2, # note the different value
                     n.components=params$n.components,
                     vertices=params$vertices,
                     disturbance=params$disturbance,
                     aggreg.factor=params$aggreg.factor,
                     planar=params$planar)
}
```

The function is then executed a sufficient number of times:

```
test2.results <- lapply(1:100, run.test2)
```

The empirical values observed for Liang Abu layers 1 and 2 (red line) can now be compared to the values measured in the simulated graph generated under the hypothesis of two initial layers. This shows, for example, that the empirical admixture value is slightly lower than the simulated admixture values:

The Journal of Open Source Software

```r
edges.res <- sapply(test2.results,
                    function(g) frag.get.parameters(g, "layer")$edges)
plot(density(edges.res), main="Edges")
abline(v=params$edges, col="red")
```

Similarly, the empirical admixture value is lower than the simulated admixture values:

```r
admix.res <- sapply(test2.results,
                    function(g) frag.layers.admixture(g, "layer"))
plot(density(admix.res), main="Admixture")
abline(v=frag.layers.admixture(abu.g12, "layer"), col="red")
```

Two functions (`frag.simul.compare` and `frag.simul.summarise`) facilitate the execution of the analytical process described above on the initial number of spatial units. The `frag.simul.compare` function takes an observed fragmentation graph, generates two series of simulated graphs corresponding to two hypotheses on the number of initial spatial units (H1 for 1 initial spatial unit and H2 for two initial spatial units), and returns a data frame of measurements made on each series (including the edge count, weights sum, balance value, disturbance value, admixture value, and cohesion values of the two spatial units).

```r
compare.res <- frag.simul.compare(abu.g12, layer.attr="layer",
                                  iter=30, summarise=FALSE)
head(compare.res$h1.data)
```

```
##   edges weightsum   balance disturbance  admixture cohesion1 cohesion2
## 1    55  278.4470 0.3472222  0.09090909 0.02384259 0.1848546 0.7913028
## 2    54  203.3956 0.2916667  0.11111111 0.03996207 0.2042161 0.7558218
## 3    51  161.7507 0.3194444  0.11764706 0.04708170 0.2217295 0.7311888
## 4    51  185.6402 0.3055556  0.05882353 0.01035035 0.4543009 0.5353487
## 5    60  281.5861 0.3333333  0.10000000 0.03112503 0.3311361 0.6377388
## 6    57  265.0727 0.3194444  0.10526316 0.03917984 0.1723818 0.7884383
```

For each of these parameters, the `frag.simul.summarise` function facilitates the comparison between empirical observed values and simulated values generated for H1 and H2.

```r
frag.simul.summarise(abu.g12, layer.attr="layer",
                     compare.res$h1.data,
                     compare.res$h2.data)
```

```
##             H1 != H2? p.value Obs. value/H1 Obs. value/H2
## edges           FALSE     0.8         lower         lower
## weightsum       FALSE    0.88        within         lower
## balance          TRUE    0.02        within         lower
## disturbance     FALSE    0.29         lower         lower
## admixture       FALSE    0.33         lower         lower
## cohesion1        TRUE       0        higher        within
## cohesion2        TRUE       0         lower        within
```

This function returns a data frame with four columns, containing, for each parameter studied:

1. whether the series of H1 values are statistically different to the H2 series (Boolean),
2. the p-value of the Wilcoxon test (numerical),

JOSS
The Journal of Open Source Software

3. whether the observed value is "within," "higher," or "lower" to the interquartile range of values for H1,
4. whether the observed value is "within," "higher," or "lower" to the interquartile range of values for H2.

Note that the `frag.simul.compare` function can optionally be set to execute and return the results of the `frag.simul.summarise` function.

Assessing spatial unit boundaries using similarity relationships

Similarity relationships are, by construction, not part of the TSAR method, which is based on the topological properties of connection networks. However, since similarity relationships are more frequent in archaeological empirical studies, the `Archeofrag` package includes various functions to handle them. This section illustrates a method to use similarity relationships using `Archeofrag` and R generic functions.

The `make_sr_graph` function takes a "frag" object and generates an `igraph` similarity network.

```
# make a frag object and generate a similarity graph:
abu.frag <- make_frag_object(sr=df.sr, fragments=fragments.info)
abu.sr <- make_sr_graph(abu.frag)
```

The `frag.relations.by.layers` function returns a table with the number of similarity relationships in and between spatial units, e.g., in the top three layers at Liang Abu:

```
# count of similarity relationships in and between layers:
simil.by.layers.df <- frag.relations.by.layers(abu.sr, "layer")
simil.by.layers.df
```

```
##
##         0    1    2
##    0   15
##    1    0  234
##    2    1   61  173
```

These values can be observed as percentages:

```
# percentage of similarity relationships in and between layers:
round(simil.by.layers.df / sum(simil.by.layers.df, na.rm=T) * 100, 0)
```

```
##
##        0   1   2
##    0   3
##    1   0  48
##    2   0  13  36
```

Considering a stratigraphic sequence, adjacent and close layers in the sequence must have lower statistical distances than distant layers. Consequently, it is expected that the result of a hierarchical clustering computed on this distance table would reflect the order of the layers. The expected result is observed for Liang Abu surface and the first two layers, suggesting an absence of significant disturbance and admixture (**??**).

```r
# turn similarity into distance:
simil.dist <- max(c(simil.by.layers.df), na.rm=T) - simil.by.layers.df
simil.dist <- as.dist(simil.dist)
# hierarchical clustering:
clust.res <- hclust(simil.dist, method="ward.D2")

library(dendextend)
clust.res$labels <- as.character(factor(clust.res$labels,
                        levels=c("0", "1", "2"),
                        labels=c("layer 0", "layer 1", "layer 2")))
dend <- as.dendrogram(clust.res)
dend %>% sort(decreasing=T) %>%
  set("labels_cex", .8)  %>%
  plot(horiz = T)
```

Characterising spatial units from their fragmentation

The second aim of the TSAR method implemented in `Archeofrag` is to characterise spatial units based on the topological properties of the connection relationships between the fragments they contain. Although this aspect is still a work in progress, some functions are already implemented and will be illustrated using simulated data. The archaeological interpretation of numerical values depends on the type of material (lithic, pottery, etc.) and the completeness or incompleteness of the objects under study and is not discussed here.

```r
# simulate a fragmentation graph:
simul.g <- frag.simul.process(initial.layers=2,
                              n.components=20,
                              vertices=70,
                              balance=.45)
# extract the subgraph of each spatial unit:
simul.g1 <- frag.get.layers(simul.g, layer.attr="layer", sel.layers="1")[[1]]
simul.g2 <- frag.get.layers(simul.g, layer.attr="layer", sel.layers="2")[[1]]
```

In a graph, a cycle is a path in which only the first and last vertices are repeated. The `frag.cycles` function searches for cycles in a graph and returns the number of cycles found for different cycle lengths. The `kmax` parameter determines the maximal length of the cycles to search for. Let us compare the cycles found in the two spatial units of the artificial graph:

```r
rbind(
  "unit1" = frag.cycles(simul.g1, kmax=5),
  "unit2" = frag.cycles(simul.g2, kmax=5))
```

```
##       3-cycles 4-cycles 5-cycles
## unit1       13        4        1
## unit2       17        9        2
```

The `frag.path.lengths` function returns the distribution of the path lengths in the graph (i.e., the number of edges between each pair of vertices). This function returns a vector whose first element is the frequency of the paths of length 1, the second element is the frequency of the paths of length 2, etc. If the `cumulative` parameter is set to `TRUE`, the function returns the cumulative relative frequency of the path lengths.

```r
frag.path.lengths(simul.g1)
```

```
## [1] 33  8  1
```

```r
frag.path.lengths(simul.g2)
```

```
## [1] 42 19  1
```

```r
frag.path.lengths(simul.g2, cumulative=T)
```

```
## [1] 1.00000000 0.45238095 0.02380952
```

In a graph, the shortest path between two vertices is the path including the least number of edges. The diameter of a graph is its longest shortest path. The `frag.diameters` function calculates the diameter of each component of the graph and returns the frequency of the values. If the `cumulative` parameter is set to `TRUE`, the function returns the cumulative relative frequency of the diameters.

```r
frag.diameters(simul.g1)
```

```
## 1 2 3
## 7 2 1
```

```r
frag.diameters(simul.g2)
```

```
## 1 2 3
## 3 6 1
```


Acknowledgements

I thank Luce Prignano, Claire Manen, Joséphine Caro, and Oliver Nakoinz for their valuable comments during the development of this package, which was finalised at the *Institut für Ur- und Frühgeschichte of Kiel*, with the support of a "Short-term research grant" from the *Deutscher Akademischer Austausch Dienst* (DAAD).

Resources

Archeofrag is available on CRAN and the code of the development version is available on Github. A Shiny application demonstrates the package.

References

Carey, V., Long, L., & Gentleman, R. (2020). *RBGL: An interface to the BOOST graph library*. https://doi.org/10.18129/B9.bioc.RBGL

Caro, J., Manen, C., Baux, A., & Plutniak, S. (2022). Les productions céramiques du Néolithique ancien et moyen: approches céramo-stratigraphique, technologique et morpho-stylistique. In C. Manen (Ed.), *Le Taï (Remoulins – Gard). Premières sociétés agropastorales du Languedoc méditerranéen (6^{e}–3^{e} millénaire avant notre ère*. Archives d'Écologie Préhistorique.

Csárdi, G., & Nepusz, T. (2006). The igraph software package for complex network research. *InterJournal*, *1695*(5), 1–9. http://igraph.org

Cziesla, E., Eickhoff, S., Arts, N., & Winter, D. (Eds.). (1990). *The big puzzle: International symposium on refitting stone artefacts*. Holos.

Frerebeau, N. (2019). Tabula: An r package for analysis, seriation, and visualization of archaeological count data. *Journal of Open Source Software*, *44*(4), 1821. https://doi.org/10.21105/joss.01821

Philippe, A., & Vibet, M.-A. (2020). Analysis of archaeological phases using the r package ArchaeoPhases. *Journal of Statistical Software*, *93*. https://doi.org/10.18637/jss.v093.c01

Plutniak, S. (2021a). *Refitting pottery fragments from the liang abu rockshelter, borneo*. Zenodo. https://doi.org/10.5281/zenodo.4719577

Plutniak, S. (2021b). The strength of parthood ties. Modelling spatial units and fragmented objects with the TSAR method – topological study of archaeological refitting. *Journal of Archaeological Science*, *136*, 105501. https://doi.org/10.1016/j.jas.2021.105501

Plutniak, S. (2022). L'analyse topologique des remontages archéologiques: la méthode TSAR et le package R *archeofrag*. *Bulletin de la Société préhistorique française*. Submitted.

Plutniak, S., Araujo, A., Puaud, S., Ferrié, J.-G., Oktaviana, A. A., Sugiyanto, B., Chazine, J.-M., & Ricaut, F.-X. (2016). Borneo as a half empty pot: Pottery assemblage from liang abu, east kalimantan. *Quaternary International*, *416*, 228–242. https://doi.org/10.1016/j.quaint.2015.11.080

R Core Team. (2020). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Schmid, C., & Serbe, B. (2017). *recexcavAAR: 3D reconstruction of archaeological excavations*. https://CRAN.R-project.org/package=recexcavAAR

Schurmans, U., & De Bie, M. (Eds.). (2007). *Fitting rocks: Lithic refitting examined*. Archaeopress. ISBN: 978-1-4073-0012-2