# Partitioned Local Depth (PaLD) Community Analyses in R

*by Lucy D'Agostino McGowan, Katherine Moore, and Kenneth Berenhaut*

**Abstract** Partitioned Local Depth (PaLD) is a framework for holistic consideration of community structure for distance-based data. This paper describes an R package, **pald**, for calculating Partitioned Local Depth (PaLD) probabilities, implementing community analyses, and creating data visualizations to display community structure. We describe how to use the package, walk through several examples, and discuss the approach in relation to commonly used techniques.

## Introduction

Partitioned Local Depth (PaLD) is a framework for holistic consideration of community structure for distance-based data. Leveraging a socially inspired perspective, the method provides network-based community information which is founded on a new measure of local depth and pairwise cohesion (partitioned local depth). The method does not require distributional assumptions, optimization criteria, nor extraneous inputs. A complete description of the perspective, together with a discussion of the underlying social motivation, theoretical results, and applications to additional data sets is provided in Berenhaut et al. (2022).

Building on existing approaches to (global) depth, local depth expresses features of centrality via an interpretable probability which is free of parameters and robust to outliers. Then, partitioning the probability which defines local depth, we obtain a measure of cohesion between pairs of individuals. Both local depth and cohesion reflect aspects of relative position (rather than absolute distance) and provide a straightforward way to account for varying density across the space. As shown in Berenhaut et al. (2022), provided that two sets are separated (in the sense that the minimum between-set distance is greater than the maximum within-set distance), cohesion is invariant under the contraction and dilation of the distances within each set. This property may be particularly valuable when one has reason to believe that there is heterogeneity in density across the space.

As cohesion captures a sense of the relationship strength between points, we can then visualize the resulting community structure with a network whose edges are weighted by (mutual) cohesion. The underlying social framework motivates a straightforward yet elegant threshold for distinguishing between strongly and weakly cohesive pairs.

Networks obtained from cohesion can be displayed using a force-directed graph drawing algorithm; here, we will graphically emphasize strong ties (colored by connected component). We refer to the connected components of the network of strong ties as community "clusters." Note that to qualify as a cluster in this definition, one may not have any strong ties with those outside the cluster, and thus the existence of disjoint groups is a strong signal for separation. Here, clusters are identified without additional user inputs nor optimization criteria. If one wishes to further break the community graph into groups, one may consider using community detection methods (such as spectral clustering or the Louvain algorithm), as available, say, in the **igraph** package. One may also use the collection of strong ties in place of (weighted) $k$-nearest neighbors in settings such as classification and smoothing. Overall, the structural information obtained from local depth, cohesion and community graphs can provide a holistic perspective to the data which does not require the use of distributional assumptions, optimization criteria nor additional user inputs.

We present a new package, **pald**, for calculating Partitioned Local Depth (PaLD) probabilities, implementing community analyses, and creating data visualizations to display community structure. This paper describes how to use the package, walks through several examples, and discusses the approach in the context of commonly used techniques, demonstrating both the novelty of the method and utility of the implementation in the package described.

## pald

The main functions in the **pald** package can be split into 3 categories:

1. A function for computing the cohesion matrix
2. Functions for extracting useful information from the cohesion matrix, such as local depths, neighbors, community clusters, and graph objects
3. Plotting functions for community graphs

In addition, the package provides a number of pertinent example data sets which may be used to demonstrate community analysis, including a synthetic data set of two-dimensional points created by Gionis et al. to demonstrate aggregation, clustering data generated from the scikit-learn Python package (Pedregosa et al., 2011), data describing cognate relationships between words across 87 Indo-European languages (Dyen et al., 1992), data compiled by Love and Irizarry (2015) of tissue gene expressions, data emplying information from the World Values Survey (Inglehart et al., 2014) on cultural values regarding family, religion, education, and institutions for several regions (Muthukrishna et al., 2020), and three example data sets generated for the Berenhaut et al. (2022) paper.

While it is not a necessity, the **pald** package is designed to function well with the pipe operator, `|>`. This functionality will be demonstrated below.

### Creating the cohesion matrix

For the purposes of the **pald** package, the sole input for the Partitioned Local Depth (PaLD) computations is a distance matrix or `dist` object. Note that the collection of input distances (or dissimilarities) does not need to satisfy the triangle inequality nor be symmetric. More generally, the method only requires triplet distance comparisons, as opposed to exact numeric distances (see Berenhaut et al. (2022)).

For demonstration purposes, we first show how one can compute a distance matrix from an input data frame with, say, two variables `x1` and `x2`. The input data may be of any dimension; in fact the PaLD framework provides advantages when considering high-dimensional data (see the **Examples** section as well as Berenhaut et al. (2022)).

```
library(pald)
df <- data.frame(
  x1 = c(6, 8, 8, 16, 4, 14),
  x2 = c(5, 4, 10, 8, 4, 10)
)
rownames(df) <- c("A", "B", "C", "D", "E", "F")
```

The `dist` function returns a (default Euclidean) pairwise distance matrix for an input data frame, as demonstrated below. If the data are already provided as a distance matrix (or `dist` object), the user can skip to the next step. Note that the distance matrix needed for the subsequent functions does not need to be a `dist` object and *need not* be symmetric.

```
d <- dist(df)
```

The function above creates a `dist` object. If converted to a matrix, this will be an $n \times n$ distance matrix, where $n$ corresponds to the number of observations in the original data frame (in this example $n = 6$).

This `dist` object, or a distance matrix, can then be passed to the `cohesion_matrix` function in order to calculate pairwise cohesion values.

Cohesion is an interpretable probability that reflects the strength of alignment of two points within local regions. It captures aspects of the relative positioning of points and accounts for varying density across the space (see Berenhaut et al. (2022)).

```
d <- dist(df)
cohesion_matrix(d)
```

```
#>            A          B          C         D          E         F
#> A 0.25000000 0.18333333 0.06666667 0.0000000 0.18333333 0.0000000
#> B 0.14000000 0.24000000 0.05000000 0.0000000 0.10666667 0.0000000
#> C 0.07333333 0.07333333 0.20333333 0.0000000 0.03333333 0.0800000
#> D 0.00000000 0.00000000 0.00000000 0.2333333 0.00000000 0.1333333
#> E 0.14000000 0.10666667 0.03333333 0.0000000 0.24000000 0.0000000
#> F 0.00000000 0.00000000 0.05000000 0.1400000 0.00000000 0.2400000
#> attr(,"class")
#> [1] "cohesion_matrix" "matrix"          "array"
```

Equivalently, the user can use the native pipe `|>` as follows.

```
df |>
  dist() |>
  cohesion_matrix()
```

```
#>            A         B         C         D         E         F
#> A 0.25000000 0.18333333 0.06666667 0.0000000 0.18333333 0.0000000
#> B 0.14000000 0.24000000 0.05000000 0.0000000 0.10666667 0.0000000
#> C 0.07333333 0.07333333 0.20333333 0.0000000 0.03333333 0.0800000
#> D 0.00000000 0.00000000 0.00000000 0.2333333 0.00000000 0.1333333
#> E 0.14000000 0.10666667 0.03333333 0.0000000 0.24000000 0.0000000
#> F 0.00000000 0.00000000 0.05000000 0.1400000 0.00000000 0.2400000
#> attr(,"class")
#> [1] "cohesion_matrix" "matrix"          "array"
```

The *cohesion matrix* output by the `cohesion_matrix` function is the main input for the majority of the remaining functions.

### Functions for extracting information from the cohesion matrix

From the *cohesion matrix*, a variety of useful quantities can be computed. Below, we create a cohesion matrix using the functions described in the previous section.

```
df |>
  dist() |>
  cohesion_matrix() -> cohesion
```

The `local_depths` function calculates the *depth* of each point, outputting a vector of local depths. Local depth is an interpretable probability which reflects aspects of relative position and centrality via distance comparisons (e.g., $d(z,x) < d(z,y)$, see Berenhaut et al. (2022)).

```
local_depths(cohesion)
```

```
#>         A         B         C         D         E         F
#> 0.6833333 0.5366667 0.4633333 0.3666667 0.5200000 0.4300000
```

In this case, the deepest point is `A`.

The `strong_threshold` function will calculate the cohesion threshold for strong ties, which reflects typical cohesion for local points (see Berenhaut et al. (2022)). Computationally, this is equal to half the average of the diagonal of the cohesion matrix, and is a threshold that may be used to distinguish between strong and weak ties.

```
strong_threshold(cohesion)
```

```
#> [1] 0.1172222
```

In this case, the threshold is a little above `0.117`.

The function `cohesion_strong` will update the cohesion matrix to set all weak ties to zero (via the `strong_threshold` function). Optionally, the matrix will also be symmetrized, using the entry-wise (parallel) minimum of the cohesion matrix and its transpose, with the default parameter `symmetric = TRUE`.

```
cohesion_strong(cohesion)
```

```
#>      A    B         C         D    E         F
#> A 0.25 0.14 0.0000000 0.0000000 0.14 0.0000000
#> B 0.14 0.24 0.0000000 0.0000000 0.00 0.0000000
#> C 0.00 0.00 0.2033333 0.0000000 0.00 0.0000000
#> D 0.00 0.00 0.0000000 0.2333333 0.00 0.1333333
#> E 0.14 0.00 0.0000000 0.0000000 0.24 0.0000000
#> F 0.00 0.00 0.0000000 0.1333333 0.00 0.2400000
#> attr(,"class")
#> [1] "cohesion_matrix" "matrix"          "array"
```

The `community_graphs` function takes the cohesion matrix and creates **igraph** objects, graphs that describe the (symmetrized) relationship structure between points. This function will output a list of three objects:

- `G`: the weighted (community) graph whose edge weights are mutual cohesion
- `G_strong`: the weighted (community) graph consisting of edges for which mutual (symmetrized) cohesion (i.e. the minimum of the two directed cohesion values for any given pair) is greater than the threshold for strong ties

- `layout`: the graph layout. By default this is provided by the Fruchterman Reingold (FR) force-directed graph drawing algorithm for the graph `G`, as implemented in the **igraph** package.

```
graphs <- community_graphs(cohesion)
graphs[["G_strong"]]
```

```
#> IGRAPH 5de0378 UNW- 6 3 --
#> + attr: name (v/c), weight (e/n)
#> + edges from 5de0378 (vertex names):
#> [1] A--B A--E D--F
```

Here we see that there are three connected components, ties `A-B` and `A-E` form the first community cluster, and the tie `D-F` which forms another.

The `any_isolated()` function will check whether there are any isolated points (according to cohesion).

```
any_isolated(cohesion)
```

Here, there are no isolated points, i.e. points having zero cohesion with all other points in the data (an extreme form of outlier).

The "community clusters" identified by PaLD are the connected components of the graph of strong ties, `G_strong`. To directly calculate these, we can use the `community_clusters` function. This will output a data frame with two columns; the first corresponds to the `point`, as identified by the row name of the original input data frame, `df`, the second identifies the `community` that the point belongs to.

```
community_clusters(cohesion)
```

```
#>   point community
#> A     A         1
#> B     B         1
#> C     C         2
#> D     D         3
#> E     E         1
#> F     F         3
```

In this example, three communities are identified with these six points. Points `A`, `B`, and `E` fall into Community 1. Point `C` is in Community 2 (a community of size 1) and points `D` and `F` fall into Community 3.

### Plotting functions

The final category of function is that for data visualization. We can begin by visualizing the points in the data frame `df` (Figure 1). When visualizing these points, it is important to have the aspect ratio of the x and y axes equal to 1 so as to not distort distances. When using the **ggplot2** package for this visualization, one can use the command `coord_fixed(ratio = 1)`. If using the `plot` function included in the base library, one can use the `asp = 1` argument.

```
library(ggplot2)
ggplot(df, aes(x1, x2)) +
  geom_text(label = rownames(df)) +
  coord_fixed(ratio = 1) +
  xlim(c(4, 16)) +
  ylim(c(4, 16))
```

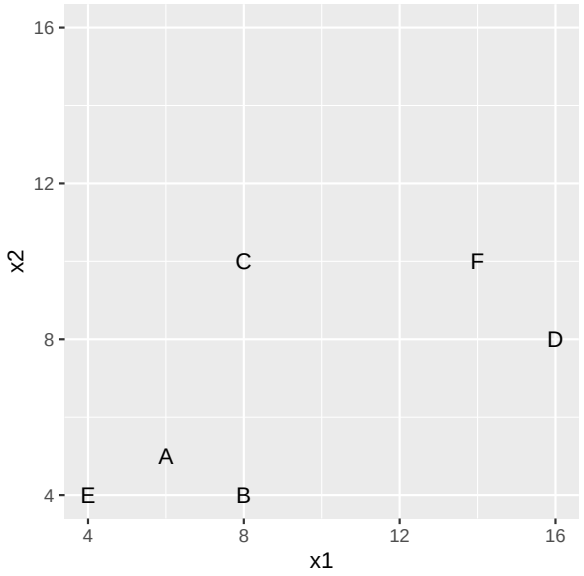


**Figure 1:** Visualization of the points from the data frame 'df'

We can pass the cohesion matrix to the `plot_community_graphs` function to view the relationship between points (Figure 2). The function will also permit parameters that can be passed to `plot.igraph` via the `...` argument.

```
plot_community_graphs(cohesion,
                      vertex.label.cex = 2,
                      vertex.label.dist = 0.9)
```

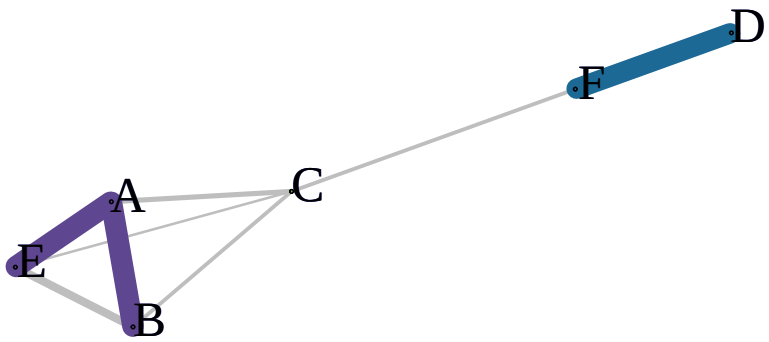


**Figure 2:** PaLD graph displaying the relationship between the points in data frame 'df'

Notice in this plot the force-directed layout does not match that of the original data frame as seen in Figure 1. Since our original data is two-dimensional, it may be reasonable to use the latter as the layout for plotting. Figure 3 includes this update as well as others addressing some of the aesthetics, such as employing more readable labels. The `layout` argument allows the user to pass a matrix to dictate the 2-dimensional layout of the graph. For example, if we wanted the graph to match the visualization displayed in Figure 1, we could pass `as.matrix(df)` (a matrix of the data frame `df`) to the `layout` argument (Figure 3). Here, we increase the vertex size and change the vertex label color, through the argument specifications `vertex.size = 100` and `vertex.label.color = "white"`. Additionally, to allow axes, we use `axes = TRUE`, and to put these back on the original scale we set `rescale = FALSE`, resetting the axis limits using `xlim` and `ylim`. The `par(pty = "s")` function forces the subsequent plot to be square.

```
par(pty = "s")

plot_community_graphs(cohesion,
                      layout = as.matrix(df),
                      vertex.size = 100,
```

```
vertex.label.color = "white",
axes = TRUE,
rescale = FALSE,
asp = 1,
xlim = c(4, 16),
ylim = c(4, 16))
```

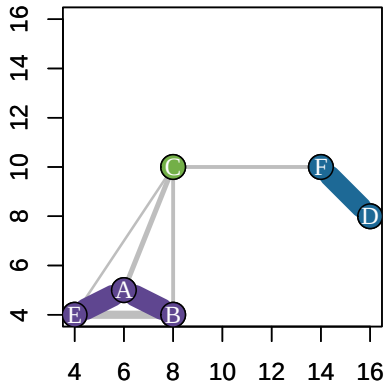


**Figure 3:** PaLD graph displaying the relationship between the points in data frame 'df', matching the original layout in Figure 1

## Examples

We will demonstrate the utility of the **pald** package through several illustrative examples.

### Community analysis for tissue gene expression data

The first example is from a subset of data from Zilliox and Irizarry (2007), McCall et al. (2011), and McCall et al. (2014), obtained from the **tissuesGeneExpression** bioconductor package (Love and Irizarry, 2015) consisting of 22,215-dimensional gene expression data from 189 tissue samples. A (Euclidean) `dist` object was created using this data set and is included in the **pald** package in an object called `tissue_dist`.

The `tissue_dist` object is a `dist` object resulting in a distance matrix with 189 rows and 189 columns.

We can create the cohesion matrix using the `cohesion_matrix` function.

```
tissue_cohesion <- cohesion_matrix(tissue_dist)
```

We can display relationships between tissue samples, both locally and globally through the `plot_community_graphs` function (Figure 4). For clarity of the display, we show how to remove the labels using `show_labels = FALSE`. We will instead color according to the labels by passing these to the `vertex.color` argument for the `plot.igraph` function (via the `...` argument). Similarly, we can add a legend using the `legend()` function, as you would for an **igraph** visualization. Additionally, we use the `edge_width_factor` and `emph_strong` arguments to adjust the width of the lines between and within PaLD communities.

```
labels <- rownames(tissue_cohesion)
plot_community_graphs(tissue_cohesion,
                      show_labels = FALSE,
                      vertex.size = 4,
                      vertex.color = as.factor(labels),
                      edge_width_factor = 35,
                      emph_strong = 5)
legend("topleft",
       legend = unique(as.factor(labels)),
       pt.bg = unique(as.factor(labels)),
       col = "black",
       pch = 21)
```

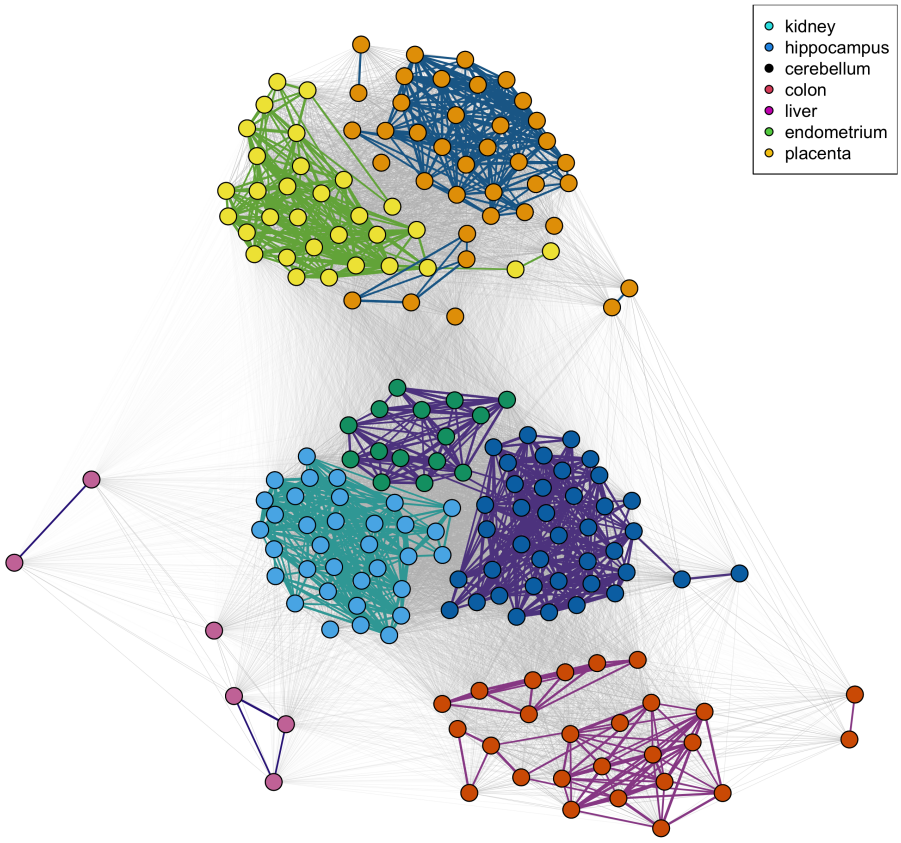


**Figure 4:** Community cluster network for the tissue data. The line colors indicate the PaLD communities; the point colors indicate the tissue classification.

A summary of community strong ties can be produced via the following code. Note that `tissue_graph_strong` is an **igraph** object corresponding to the graph displayed in Figure 4, restricted only to strong ties.

```
tissue_graphs <- community_graphs(tissue_cohesion)
tissue_graph_strong <- tissue_graphs[["G_strong"]]
E <- igraph::get.edgelist(tissue_graph_strong)
tissue_strong_ties <- data.frame(
  strong_ties = apply(E, 1, paste, collapse = ",")
)
tissue_strong_ties |>
  dplyr::count(strong_ties) |>
  dplyr::arrange(-n)
```

```
#>               strong_ties   n
#> 1           kidney,kidney 273
#> 2             colon,colon 241
#> 3 hippocampus,hippocampus 191
#> 4   cerebellum,cerebellum 186
#> 5             liver,liver  85
#> 6 endometrium,endometrium  81
#> 7       placenta,placenta   4
#> 8      kidney,endometrium   3
```

Note that there are only three strong ties between different tissue types (kidney and endometrium) in the community cluster network of 1,064 strong ties total.

### Cognate-based Language Families

This example explores a data set from Dyen et al. (1992) that summarizes relationships between 87 Indo-European languages from the perspective of cognates, coded using 2,655-dimensional binary

vectors. A `dist` object was created from this data set and is included in the **pald** package in an object called `cognate_dist`.

Here we will demonstrate how one can further apply functions in the **igraph** package to objects output from the **pald** package. We can first use the `cohesion_matrix` function to calculate the cohesion matrix and the `community_graphs` function to create a list with the weighted community graph, the weighted community graph with only strong ties included, and the layout. From this, we can extract the graph with only the strong ties, here called `cognate_graph_strong`.

```
cognate_cohesion <- cohesion_matrix(cognate_dist)
cognate_graphs <- community_graphs(cognate_cohesion)

cognate_graph_strong <- cognate_graphs[["G_strong"]]
```

We can then use the `neighbors` function from the **igraph** package to extract the strong neighbors in this graph. For example, we can extract all neighbors for the language "French", via the following code.

```
french_neighbors <- igraph::neighbors(cognate_graph_strong, "French")
french_neighbors
```

```
#> + 8/87 vertices, named, from b09505b:
#> [1] Italian         Ladin           Provencal       Walloon
#> [5] French_Creole_C French_Creole_D Spanish         Catalan
```

Similarly, we can sort and print the associated neighborhood weights by subsetting the cohesion matrix.

```
cognate_cohesion["French", french_neighbors] |>
  sort(decreasing = TRUE)
```

```
#>         Walloon       Provencal French_Creole_C French_Creole_D           Ladin
#>      0.03258771      0.02871174      0.02406057      0.02406057      0.02094596
#>         Italian         Catalan         Spanish
#>      0.01997696      0.01859688      0.01679733
```

We can again use the `plot_community_graphs` function to visualize the community clusters (Figure 5). One may note the commonly identifiable language clusters and that, under a slight rotation, some of the underlying geography is mirrored in the plot.

```
plot_community_graphs(
  cognate_cohesion,
  edge_width_factor = 30,
  emph_strong = 3,
  vertex.size = 3,
  vertex.label.cex = 0.7,
  vertex.label.dist = 1
)
```

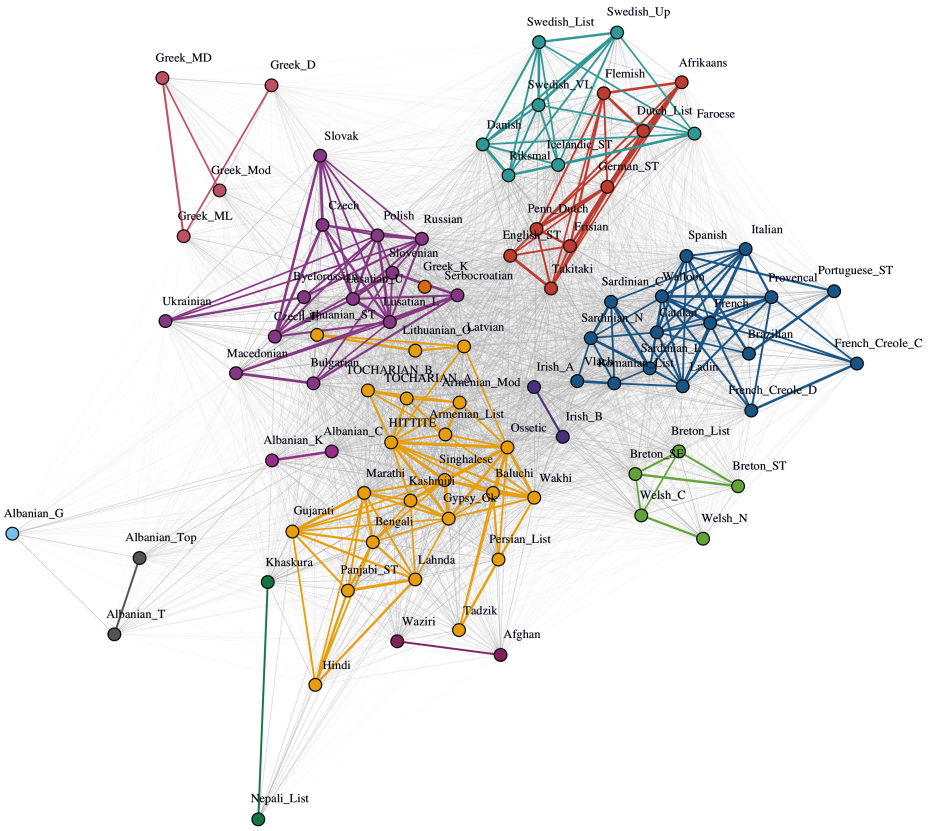


**Figure 5:** Community structure for 87 Indo-European languages, which employs cognate information that was coded via 2,665-dimensional binary vectors. Commonly identifiable language clusters arise along with informative inter- and intra-cluster structure. Several ancient languages are centrally located.

### Community analysis for generated data

The **pald** package includes three randomly generated data frames corresponding to plots from Berenhaut et al. (2022):

- `exdata1` is a data set consisting of 8 points used to create Figure 1 in Berenhaut et al. (2022)
- `exdata2` is a data set consisting of 16 points used to create Figure 2 in Berenhaut et al. (2022)
- `exdata3` is a data set consisting of 240 points used to create Figure 4D in Berenhaut et al. (2022)

Here, we will demonstrate how to use `exdata3`. These points were generated from bivariate normal distributions with varying means and variances. There are eight "true" communities.

We will contrast community analysis via PaLD to standard cluster analysis by considering two common clustering methods. The code below calculates the cohesion matrix (`exdata_cohesion`) as well as the community clusters obtained via PaLD (`exdata_pald`), along with $k = 8$ clusters obtained via $k$-means (`exdata_kmeans`) and hierarchical clustering using complete linkage (`exdata_hclust`). Recall that there is no need to determine values for extraneous inputs in the case of PaLD (e.g. the number of clusters, as is necessary to specify for $k$-means and hierarchical clustering).

```
exdata_cohesion <- exdata3 |>
  dist() |>
  cohesion_matrix()

exdata_pald <- community_clusters(exdata_cohesion)$community

exdata_kmeans <- kmeans(exdata3, 8)$cluster

exdata_hclust <- exdata3 |>
```

```
  dist() |>
  hclust() |>
  cutree(k = 8)
```

The information is displayed in Figure 6.

```
par(mfrow = c(1, 3), pty = "s")
plot(
  exdata3,
  pch = 16,
  col = pald_colors[exdata_pald],
  xlab = "",
  ylab = "",
  main = "PaLD Communities",
  asp = 1
)
plot(
  exdata3,
  pch = 16,
  col = pald_colors[exdata_kmeans],
  xlab = "",
  ylab = "",
  main = "K-Means Clusters (k = 8)",
  asp = 1
)
plot(
  exdata3,
  pch = 16,
  col = pald_colors[exdata_hclust],
  xlab = "",
  ylab = "",
  main = "Hiearchical Clusters (k = 8)",
  asp = 1
)
```

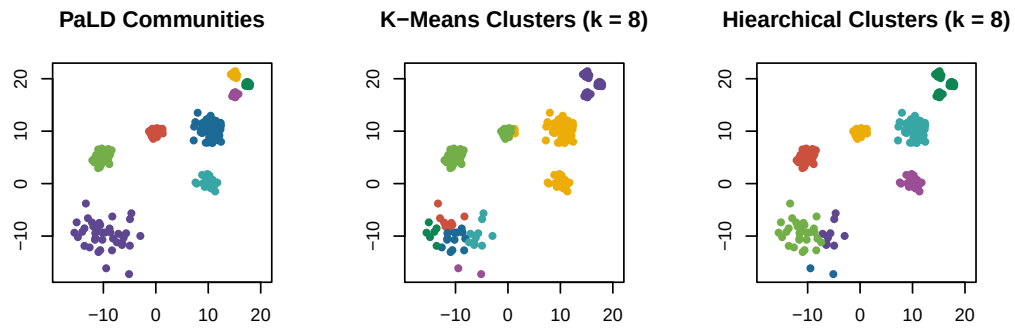


**Figure 6:** PaLD, k-means, and hierarchical 8-clustering of randomly generated example data (from Berenhaut et al. (2022); Figure 4D).

Cohesion is particularly useful when considering data with varying local density; see Berenhaut et al. (2022) for further examples, discussion, and theoretical results. Note that the PaLD algorithm is able to detect the eight natural groups within the data (along with inter- and intra-community structure not displayed here) without the use of any additional inputs (e.g., number of clusters) nor optimization criteria. Despite the user input of the "correct" number of clusters (i.e., $k = 8$) both $k$-means and hierarchical clustering do not provide the desired result.

### Cultural and Pyschological distance analysis

In this example we perform a PaLD analysis for cultural distances obtained in Muthukrishna et al. (2020) from two recent waves of the World Values Survey (2005 to 2009 and 2010 to 2014; see Inglehart et al. (2014)). Distances are computed using the cultural fixation index (CFST), which is a measure built

on the framework of fixation indices from population biology (Bell et al. (2009); Cavalli-Sforza et al. (1994)). Recall that the foundation of PaLD in within-triplet comparisons allows for the employment of application-dependent and non-Euclidean measures of dissimilarity. The `dist` object is included in the **pald** package (`cultures`). We will first create the cohesion matrix using the `dist` object `cultures`, and proceed to plot the community graph.

```
cultures_cohesion <- cohesion_matrix(cultures)

plot_community_graphs(
  cultures_cohesion,
  edge_width_factor = 30,
  emph_strong = 3,
  vertex.label.cex = 0.7,
  vertex.size = 3,
  vertex.label.dist = 1
)
```

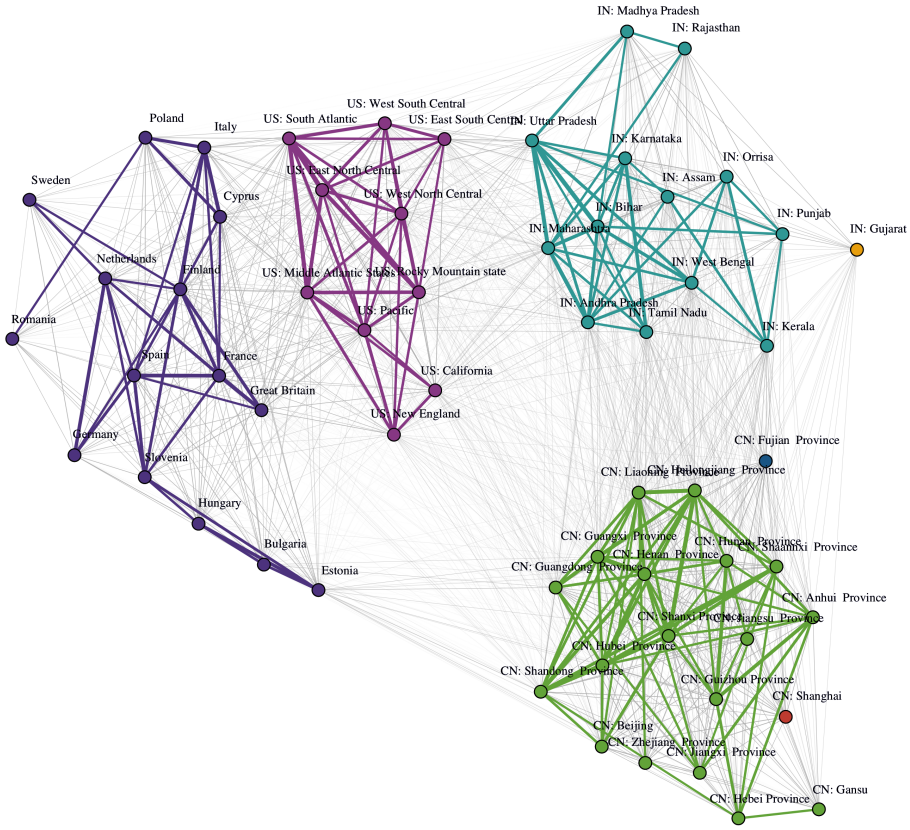


**Figure 7:** Community structure for cultural distance data.

In addition to viewing the local and global community structure as seen in Figure 7, the **pald** package allows for a two-dimensional display of cohesion against distance for the data, via the `dist_cohesion_plot` function, as seen below (Figure 8).

```
dist_cohesion_plot(cultures, mutual = TRUE)
```

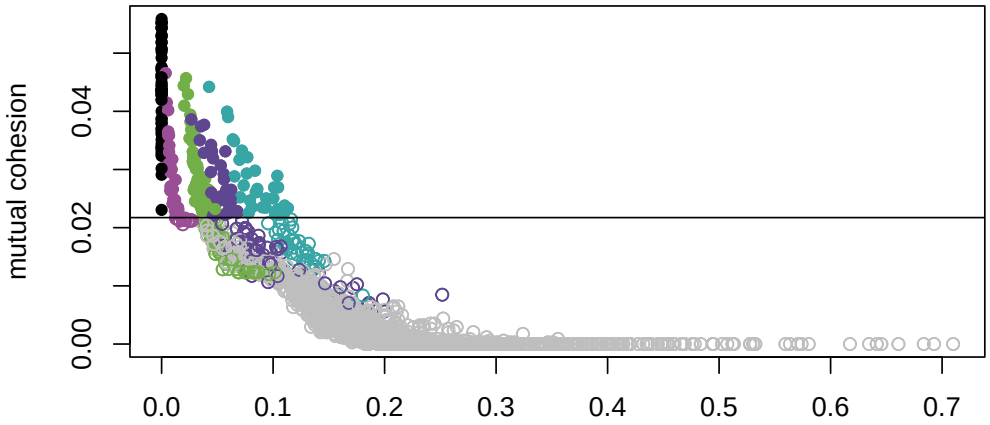


**Figure 8:** A plot of cohesion versus distance for the data. The identified communities are colored as in Figure 7.

Notice here that the magnitude of the distances within each of the identified communities varies substantially between regions; in fact, the most disparate two regions in the United States (at distance $\approx 0.027$) are far closer than the two most similar in India (at distance $\approx 0.043$). Despite this, India remains a cohesive whole, and locally disparate regions in the United States such as East South Central and California are not strongly cohesive. For discussion of subtleties in local density (see Berenhaut et al. (2022). Additionally, currently available techniques require specification of parameters, as seen in the previous section. For further discussion of the cultural distance data in relation to community analysis see Berenhaut et al. (2022).

### Summary

This paper introduces the **pald** package, demonstrating its utility for providing novel parameter-free community analysis which can easily be implemented for a variety of data sets, supplementing results from other methods for clustering, embedding, nearest neighbors, etc. Example code is provided along with discussion in the context of other commonly used R-based approaches.

## Bibliography

A. V. Bell, P. J. Richerson, and R. McElreath. Culture rather than genes provides greater scope for the evolution of large-scale human prosociality. *Proceedings of the National Academy of Sciences*, 106(42): 17671–17674, 2009. [p11]

K. S. Berenhaut, K. E. Moore, and R. L. Melvin. A social perspective on perceived distances reveals deep community structure. *Proceedings of the National Academy of Sciences*, 119(4), 2022. [p1, 2, 3, 9, 10, 12]

L. L. Cavalli-Sforza, L. Cavalli-Sforza, P. Menozzi, and A. Piazza. *The history and geography of human genes*. Princeton university press, 1994. [p11]

I. Dyen, J. B. Kruskal, and P. Black. An indoeuropean classification: A lexicostatistical experiment. *Transactions of the American Philosophical Society*, 82(5):iii, 1992. doi: 10.2307/1006517. [p2, 7]

A. Gionis, H. Mannila, and P. Tsaparas. Clustering aggregation. *ACM Transactions on Knowledge Discovery from Data*, 1:1–30. [p2]

R. Inglehart, C. Haerpfer, A. Moreno, C. Welzel, K. Kizilova, J. Díez-Medrano, M. Lagos, P. Norris, E. Ponarin, B. Puranen, et al. World values survey: All rounds–country-pooled datafile 1981-2014. madrid: Jd systems institute, 2014. 2014. [p2, 10]

M. Love and R. Irizarry. *tissuesGeneExpression. Subset of gene expression data*, 2015. URL `https://github.com/genomicsclass/tissuesGeneExpression`. [p2, 6]

M. N. McCall, K. Uppal, H. A. Jaffee, M. J. Zilliox, and R. A. Irizarry. The gene expression barcode: leveraging public data repositories to begin cataloging the human and murine transcriptomes. *Nucleic Acids Research*, 39(suppl_1):D1011–D1015, 2011. [p6]

M. N. McCall, H. A. Jaffee, S. J. Zelisko, N. Sinha, G. Hooiveld, R. A. Irizarry, and M. J. Zilliox. The gene expression barcode 3.0: improved data processing and mining tools. *Nucleic Acids Research*, 42 (D1):D938–D943, 2014. [p6]

M. Muthukrishna, A. V. Bell, J. Henrich, C. M. Curtin, A. Gedranovich, J. McInerney, and B. Thue. Beyond western, educated, industrial, rich, and democratic (weird) psychology: Measuring and mapping scales of cultural and psychological distance. *Psychological Science*, 31(6):678–701, 2020. [p2, 10]

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011. [p2]

M. J. Zilliox and R. A. Irizarry. A gene expression bar code for microarray data. *Nature Methods*, 4(11): 911–913, 2007. [p6]

*Lucy D'Agostino McGowan*
*Wake Forest University*
*Winston-Salem, NC*
*27106*
mcgowald@wfu.edu

*Katherine Moore*
*Amherst College*
*Amherst, MA*
*1002*
kmoore@amherst.edu

*Kenneth Berenhaut*
*Wake Forest University*
*Winston-Salem, NC*
*27106*
berenhks@wfu.edu