Genie: A New, Fast, and Outlier-Resistant Hierarchical Clustering Algorithm and Its R Interface

Marek Gągolewski

marek@gagolewski.com

Systems Research Institute, Polish Academy of Sciences
Faculty of Mathematics and Information Science, Warsaw University of Technology
Data Science Retreat, Berlin

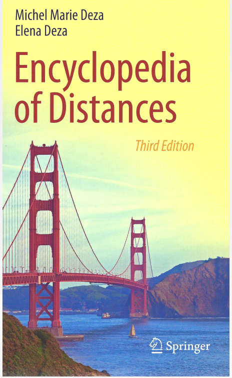
European R Users Meeting; Poznań, Poland, 2016

Outline

- Hierarchical Agglomerative Clustering – Issues
- The Genie Algorithm
- Implementation
- A Possible Generalization

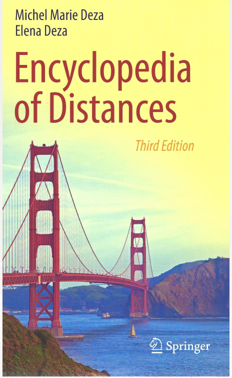
Inputs

- Let $\mathfrak{d}$ be a pairwise dissimilarity measure over $\mathcal{X}$ (**distance**, see Deza and Deza, 2014), i.e., $\mathfrak{d} : \mathcal{X} \times \mathcal{X} \to [0, \infty]$ *at least* such that:
  - $\mathfrak{d}(\mathbf{x}', \mathbf{x}'') = \mathfrak{d}(\mathbf{x}'', \mathbf{x}')$ for all $\mathbf{x}', \mathbf{x}'' \in \mathcal{X}$,
  - $\mathfrak{d}(\mathbf{x}', \mathbf{x}'') = 0$ if and only if $\mathbf{x}' = \mathbf{x}''$.



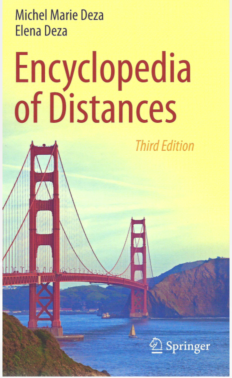
Inputs

- Let $\mathfrak{d}$ be a pairwise dissimilarity measure over $\mathcal{X}$ (**distance**, see Deza and Deza, 2014), i.e., $\mathfrak{d}: \mathcal{X} \times \mathcal{X} \to [0, \infty]$ *at least* such that:
    - $\mathfrak{d}(\mathbf{x}', \mathbf{x}'') = \mathfrak{d}(\mathbf{x}'', \mathbf{x}')$ for all $\mathbf{x}', \mathbf{x}'' \in \mathcal{X}$,
    - $\mathfrak{d}(\mathbf{x}', \mathbf{x}'') = 0$ if and only if $\mathbf{x}' = \mathbf{x}''$.
- Some examples:
    - $\mathbb{R}^d$ with the Euclidean or Manhattan metric,
    - $\{A, C, T, G\}^*$ with the Levenshtein or Dinu rank distance,
    - a chain with the natural distance,
    - raster images with RMSE of brightness values,
    - a Cartesian product of the above spaces with a convex combination of different distances.



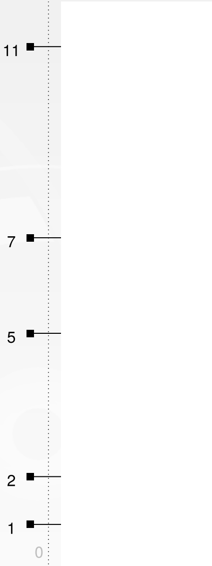
Inputs

- Let $\mathfrak{d}$ be a pairwise dissimilarity measure over $\mathcal{X}$ (**distance**, see Deza and Deza, 2014), i.e., $\mathfrak{d} : \mathcal{X} \times \mathcal{X} \to [0, \infty]$ *at least* such that:
  - $\mathfrak{d}(\mathbf{x}', \mathbf{x}'') = \mathfrak{d}(\mathbf{x}'', \mathbf{x}')$ for all $\mathbf{x}', \mathbf{x}'' \in \mathcal{X}$,
  - $\mathfrak{d}(\mathbf{x}', \mathbf{x}'') = 0$ if and only if $\mathbf{x}' = \mathbf{x}''$.
- Some examples:
  - $\mathbb{R}^d$ with the Euclidean or Manhattan metric,
  - $\{A, C, T, G\}^*$ with the Levenshtein or Dinu rank distance,
  - a chain with the natural distance,
  - raster images with RMSE of brightness values,
  - a Cartesian product of the above spaces with a convex combination of different distances.
- Let $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}\} \subseteq \mathcal{X}$ be a set of n objects.



Hierarchical Agglomerative Clustering

- Initially, $\mathcal{C}^{(0)} = \{\{\mathbf{x}^{(1)}\}, \ldots, \{\mathbf{x}^{(n)}\}\}$.



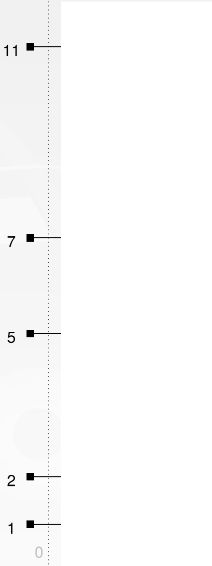
Hierarchical Agglomerative Clustering

- Initially, $\mathcal{C}^{(0)} = \{\{\mathbf{x}^{(1)}\}, \dots, \{\mathbf{x}^{(n)}\}\}$.
- There are $n-j$ clusters at the j-th step of the procedure: $\mathcal{C}^{(j)} = \{C_1^{(j)}, \dots, C_{n-j}^{(j)}\}$ is a partition of $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}$.

  When proceeding from step $j-1$ to j, we *decide* which $C_u^{(j-1)}$ and $C_v^{(j-1)}$, $u < v$, are to be merged so that we get:

  - $C_i^{(j)} = C_i^{(j-1)}$ for $u \neq i < v$;
  - $C_u^{(j)} = C_u^{(j-1)} \cup C_v^{(j-1)}$;
  - $C_i^{(j)} = C_{i+1}^{(j-1)}$ for $i > v$.

  Thus, $(\mathcal{C}^{(0)}, \mathcal{C}^{(1)}, \dots, \mathcal{C}^{(n-1)})$ is a sequence of nested partitions with $\mathcal{C}^{(n-1)} = \{\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}\}$.



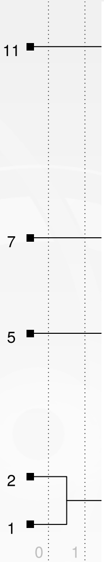
Hierarchical Agglomerative Clustering

- Initially, $\mathcal{C}^{(0)} = \{\{\mathbf{x}^{(1)}\}, \dots, \{\mathbf{x}^{(n)}\}\}$.
- There are $n - j$ clusters at the j-th step of the procedure: $\mathcal{C}^{(j)} = \{C_1^{(j)}, \dots, C_{n-j}^{(j)}\}$ is a partition of $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}$.

  When proceeding from step $j-1$ to j,
  we *decide* which $C_u^{(j-1)}$ and $C_v^{(j-1)}$, $u < v$,
  are to be merged so that we get:

  - $C_i^{(j)} = C_i^{(j-1)}$ for $u \neq i < v$;
  - $C_u^{(j)} = C_u^{(j-1)} \cup C_v^{(j-1)}$;
  - $C_i^{(j)} = C_{i+1}^{(j-1)}$ for $i > v$.

  Thus, $(\mathcal{C}^{(0)}, \mathcal{C}^{(1)}, \dots, \mathcal{C}^{(n-1)})$ *is a sequence of nested partitions with* $\mathcal{C}^{(n-1)} = \{\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}\}$.

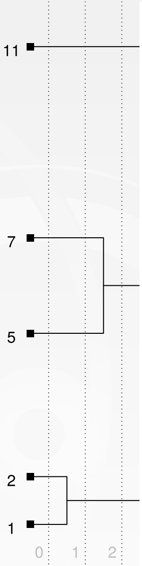


Hierarchical Agglomerative Clustering

- Initially, $\mathcal{C}^{(0)} = \{\{\mathbf{x}^{(1)}\}, \dots, \{\mathbf{x}^{(n)}\}\}$.
- There are $n - j$ clusters at the j-th step of the procedure: $\mathcal{C}^{(j)} = \{C_1^{(j)}, \dots, C_{n-j}^{(j)}\}$ is a partition of $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}$.

  When proceeding from step $j-1$ to j, we *decide* which $C_u^{(j-1)}$ and $C_v^{(j-1)}$, $u < v$, are to be merged so that we get:
  - $C_i^{(j)} = C_i^{(j-1)}$ for $u \neq i < v$;
  - $C_u^{(j)} = C_u^{(j-1)} \cup C_v^{(j-1)}$;
  - $C_i^{(j)} = C_{i+1}^{(j-1)}$ for $i > v$.

  Thus, $(\mathcal{C}^{(0)}, \mathcal{C}^{(1)}, \dots, \mathcal{C}^{(n-1)})$ *is a sequence of nested partitions with* $\mathcal{C}^{(n-1)} = \{\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}\}$.

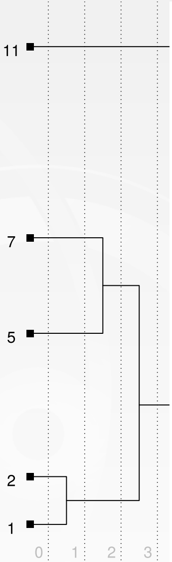


Hierarchical Agglomerative Clustering

- Initially, $\mathcal{C}^{(0)} = \{\{\mathbf{x}^{(1)}\}, \dots, \{\mathbf{x}^{(n)}\}\}$.
- There are $n-j$ clusters at the j-th step of the procedure: $\mathcal{C}^{(j)} = \{C_1^{(j)}, \dots, C_{n-j}^{(j)}\}$ is a partition of $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}$.

  When proceeding from step $j-1$ to j, we *decide* which $C_u^{(j-1)}$ and $C_v^{(j-1)}$, $u < v$, are to be merged so that we get:
  - $C_i^{(j)} = C_i^{(j-1)}$ for $u \neq i < v$;
  - $C_u^{(j)} = C_u^{(j-1)} \cup C_v^{(j-1)}$;
  - $C_i^{(j)} = C_{i+1}^{(j-1)}$ for $i > v$.

  Thus, $(\mathcal{C}^{(0)}, \mathcal{C}^{(1)}, \dots, \mathcal{C}^{(n-1)})$ is a sequence of nested partitions with $\mathcal{C}^{(n-1)} = \{\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}\}$.



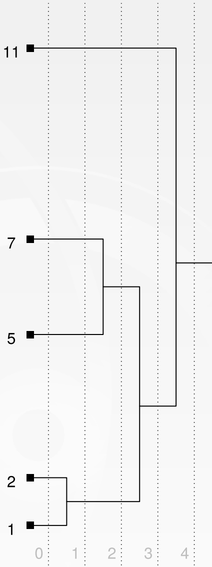
Hierarchical Agglomerative Clustering

- Initially, $\mathcal{C}^{(0)} = \{\{\mathbf{x}^{(1)}\}, \dots, \{\mathbf{x}^{(n)}\}\}$.
- There are $n - j$ clusters at the j-th step of the procedure: $\mathcal{C}^{(j)} = \{C_1^{(j)}, \dots, C_{n-j}^{(j)}\}$ is a partition of $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}$.

  When proceeding from step $j-1$ to j, we *decide* which $C_u^{(j-1)}$ and $C_v^{(j-1)}$, $u < v$, are to be merged so that we get:

  - $C_i^{(j)} = C_i^{(j-1)}$ for $u \neq i < v$;
  - $C_u^{(j)} = C_u^{(j-1)} \cup C_v^{(j-1)}$;
  - $C_i^{(j)} = C_{i+1}^{(j-1)}$ for $i > v$.

  Thus, $(\mathcal{C}^{(0)}, \mathcal{C}^{(1)}, \dots, \mathcal{C}^{(n-1)})$ is a sequence of nested partitions with $\mathcal{C}^{(n-1)} = \{\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}\}$.



Hierarchical Agglomerative Clustering

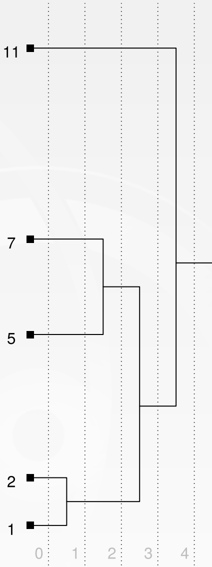
- Initially, $\mathcal{C}^{(0)} = \{\{\mathbf{x}^{(1)}\}, \dots, \{\mathbf{x}^{(n)}\}\}$.
- There are $n-j$ clusters at the j-th step of the procedure: $\mathcal{C}^{(j)} = \{C_1^{(j)}, \dots, C_{n-j}^{(j)}\}$ is a partition of $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}$.

  When proceeding from step $j-1$ to j, we *decide* which $C_u^{(j-1)}$ and $C_v^{(j-1)}$, $u < v$, are to be merged so that we get:

  - $C_i^{(j)} = C_i^{(j-1)}$ for $u \neq i < v$;
  - $C_u^{(j)} = C_u^{(j-1)} \cup C_v^{(j-1)}$;
  - $C_i^{(j)} = C_{i+1}^{(j-1)}$ for $i > v$.

  Thus, $(\mathcal{C}^{(0)}, \mathcal{C}^{(1)}, \dots, \mathcal{C}^{(n-1)})$ is a sequence of nested partitions with $\mathcal{C}^{(n-1)} = \{\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}\}$.
- Classical linkage criterion – for some *extension* $\tilde{\mathfrak{d}}$ of $\mathfrak{d}$ to $2^{\mathcal{X}}$:

$$\arg\min_{(u,v), u<v} \tilde{\mathfrak{d}}(C_u^{(j-1)}, C_v^{(j-1)}).$$



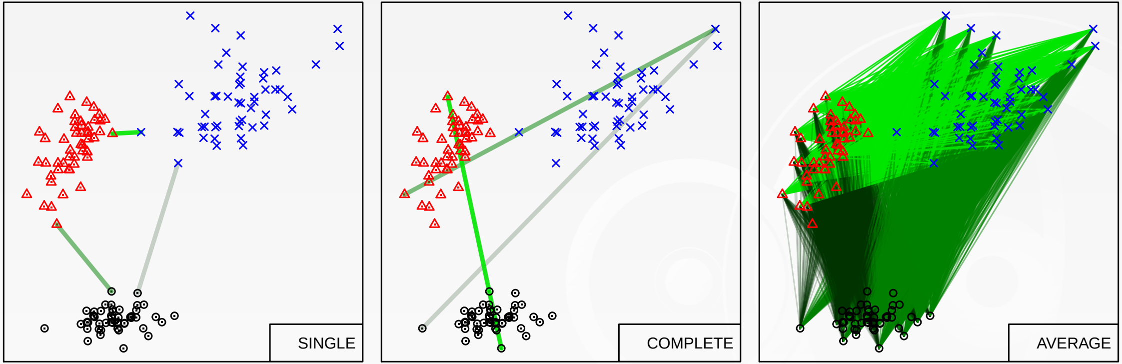
Hierarchical Agglomerative Clustering

$$\tilde{\mathfrak{d}}(C_u, C_v) = \ldots$$

$$\min_{\mathbf{a}\in C_u, \mathbf{b}\in C_v} \mathfrak{d}(\mathbf{a}, \mathbf{b})$$

$$\max_{\mathbf{a}\in C_u, \mathbf{b}\in C_v} \mathfrak{d}(\mathbf{a}, \mathbf{b})$$

$$\frac{1}{|C_u||C_v|} \sum_{\mathbf{a}\in C_u, \mathbf{b}\in C_v} \mathfrak{d}(\mathbf{a}, \mathbf{b})$$



Hierarchical Agglomerative Clustering
Some Issues

Hierarchical clustering is generally computationally demanding, but there are some relatively efficient algorithms for the single linkage (MST).

Hierarchical Agglomerative Clustering

Some Issues

Hierarchical clustering is generally computationally demanding, but there are some relatively efficient algorithms for the single linkage (MST).

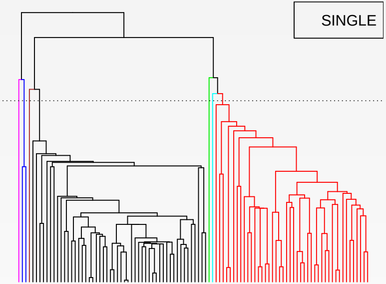
Some linkages (the single one in particular) are very sensitive to outliers, produce highly skewed dendrograms, and often don't reflect the underlying data structure.

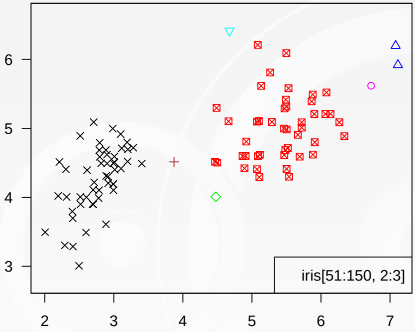
Hierarchical Agglomerative Clustering

Some Issues

Hierarchical clustering is generally computationally demanding, but there are some relatively efficient algorithms for the single linkage (MST).

Some linkages (the single one in particular) are very sensitive to outliers, produce highly skewed dendrograms, and often don't reflect the underlying data structure.





Is it a clustering or an outlier detection algorithm?

Hierarchical Agglomerative Clustering
Inequity Measures

In order to quantify the *skewness* of a partition,
we may refer to the notion of an inequity (poverty) index.

Hierarchical Agglomerative Clustering
Inequity Measures

In order to quantify the *skewness* of a partition,
we may refer to the notion of an inequity (poverty) index.

Let $\mathcal{G} = \{(x_1, \ldots, x_n) : x_1 \geq \cdots \geq x_n \geq 0\}$.

Hierarchical Agglomerative Clustering

Inequity Measures

In order to quantify the *skewness* of a partition,
we may refer to the notion of an inequity (poverty) index.

Let $\mathcal{G} = \{(x_1, \ldots, x_n) : x_1 \geq \cdots \geq x_n \geq 0\}$.

Definition.

$\mathsf{F} : \mathcal{G} \to [0, 1]$ is an **inequity measure** (see Aristondo, García-Lapresta, Lasso de la Vega, Marques Pereira, 2013; Beliakov, Gagolewski, James, 2016), whenever:

- $\inf_{\mathbf{x}\in\mathcal{G}} \mathsf{F}(\mathbf{x}) = 0$ and $\sup_{\mathbf{x}\in\mathcal{G}} \mathsf{F}(\mathbf{x}) = 1$,
- for any $\mathbf{x}, \mathbf{y} \in \mathcal{G}$, $\sum_{i=1}^n x_i = \sum_{i=1}^n y_i$ if for all $i = 1, \ldots, n$ it holds $\sum_{j=1}^i x_j \leq \sum_{j=1}^i y_j$, then $\mathsf{F}(\mathbf{x}) \leq \mathsf{F}(\mathbf{y})$. *(Schur-convexity)*

Hierarchical Agglomerative Clustering
Inequity Measures

In order to quantify the *skewness* of a partition,
we may refer to the notion of an inequity (poverty) index.

Let $\mathcal{G} = \{(x_1, \ldots, x_n) : x_1 \geq \cdots \geq x_n \geq 0\}$.

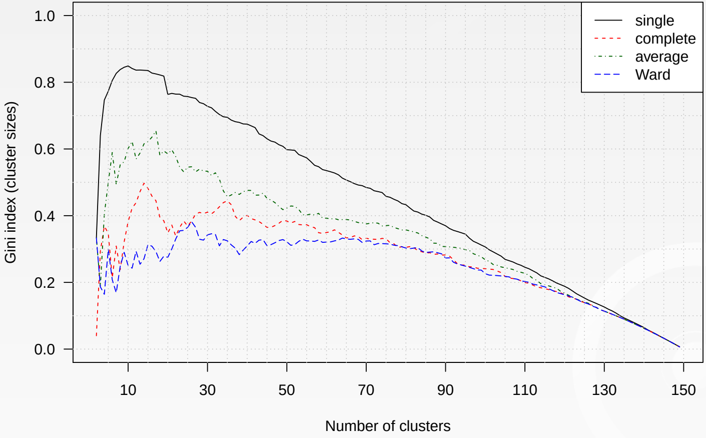
Definition.

$\mathsf{F} : \mathcal{G} \to [0, 1]$ is an **inequity measure** (see Aristondo, García-Lapresta, Lasso de la Vega, Marques Pereira, 2013; Beliakov, Gagolewski, James, 2016), whenever:

- $\inf_{\mathbf{x}\in\mathcal{G}} \mathsf{F}(\mathbf{x}) = 0$ and $\sup_{\mathbf{x}\in\mathcal{G}} \mathsf{F}(\mathbf{x}) = 1$,
- for any $\mathbf{x}, \mathbf{y} \in \mathcal{G}$, $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$ if for all $i = 1, \ldots, n$ it holds $\sum_{j=1}^{i} x_j \leq \sum_{j=1}^{i} y_j$, then $\mathsf{F}(\mathbf{x}) \leq \mathsf{F}(\mathbf{y})$. *(Schur-convexity)*

Examples: the normalized Gini, Bonferroni, de Vergottini indices.

Hierarchical Agglomerative Clustering
Inequity Measures



The Gini indices for the cluster size distributions – *Iris* dataset

$$\mathsf{G}(\mathbf{c}) = \frac{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}|c_i - c_j|}{(n-1)\sum_{i=1}^{n} c_i},$$

$$\mathbf{c} = \left(|C_1^{(j)}|, \ldots, |C_{n-j}^{(j)}|\right)$$

The Genie Algorithm

Gagolewski M., Bartoszuk M., Cena A., Genie: A new, fast, and outlier-resistant hierarchical clustering algorithm, *Information Sciences* 363, 2016, 8–23;
Gagolewski M., Cena A., Bartoszuk M., Hierarchical Clustering via Penalty-Based Aggregation and the Genie Approach, *Lecture Notes in Artificial Intelligence* 9880 (MDAI'16), 2016, 191–202.

In order to compensate the drawbacks of the single linkage scheme, while retaining its simplicity, we proposed the Genie linkage scheme.

The Genie Algorithm

Gagolewski M., Bartoszuk M., Cena A., Genie: A new, fast, and outlier-resistant hierarchical clustering algorithm, *Information Sciences* 363, 2016, 8–23;
Gagolewski M., Cena A., Bartoszuk M., Hierarchical Clustering via Penalty-Based Aggregation and the Genie Approach, *Lecture Notes in Artificial Intelligence* 9880 (MDAI'16), 2016, 191–202.

In order to compensate the drawbacks of the single linkage scheme, while retaining its simplicity, we proposed the Genie linkage scheme.

Let F be a fixed inequity measure and $g \in (0, 1]$ be some threshold.
Under the assumption that $c_i = |C_i^{(j)}|$, at the j-th step:

The Genie Algorithm

Gagolewski M., Bartoszuk M., Cena A., Genie: A new, fast, and outlier-resistant hierarchical clustering algorithm, *Information Sciences* 363, 2016, 8–23;
Gagolewski M., Cena A., Bartoszuk M., Hierarchical Clustering via Penalty-Based Aggregation and the Genie Approach, *Lecture Notes in Artificial Intelligence* 9880 (MDAI'16), 2016, 191–202.

In order to compensate the drawbacks of the single linkage scheme, while retaining its simplicity, we proposed the Genie linkage scheme.

Let F be a fixed inequity measure and $g \in (0,1]$ be some threshold.
Under the assumption that $c_i = |C_i^{(j)}|$, at the j-th step:

(a) If $\mathsf{F}(c_{(n-j)}, \dots, c_{(1)}) \le g$,

$$\arg\min_{(u,v),u<v} \left(\min_{\mathbf{a}\in C_u^{(j)},\mathbf{b}\in C_v^{(j)}} \mathfrak{d}(\mathbf{a},\mathbf{b}) \right),$$

(the single linkage criterion)

The Genie Algorithm

Gagolewski M., Bartoszuk M., Cena A., Genie: A new, fast, and outlier-resistant hierarchical clustering algorithm, *Information Sciences* 363, 2016, 8–23;
Gagolewski M., Cena A., Bartoszuk M., Hierarchical Clustering via Penalty-Based Aggregation and the Genie Approach, *Lecture Notes in Artificial Intelligence* 9880 (MDAI'16), 2016, 191–202.

In order to compensate the drawbacks of the single linkage scheme, while retaining its simplicity, we proposed the Genie linkage scheme.

Let F be a fixed inequity measure and $g \in (0,1]$ be some threshold.
Under the assumption that $c_i = |C_i^{(j)}|$, at the j-th step:

(a) If $\mathsf{F}(c_{(n-j)}, \dots, c_{(1)}) \le g$,

$$\arg \min_{(u,v), u<v} \left(\min_{\mathbf{a} \in C_u^{(j)}, \mathbf{b} \in C_v^{(j)}} \mathfrak{d}(\mathbf{a}, \mathbf{b}) \right),$$

(the single linkage criterion)

(b) If $\mathsf{F}(c_{(n-j)}, \dots, c_{(1)}) > g$,

$$\arg \min_{\substack{(u,v), u<v, \\ c_u = c_{(1)} \text{ or } \\ c_v = c_{(1)}}} \left(\min_{\mathbf{a} \in C_u^{(j)}, \mathbf{b} \in C_v^{(j)}} \mathfrak{d}(\mathbf{a}, \mathbf{b}) \right),$$

(search domain restricted to pairs of clusters s.t. one of them is of the smallest size)

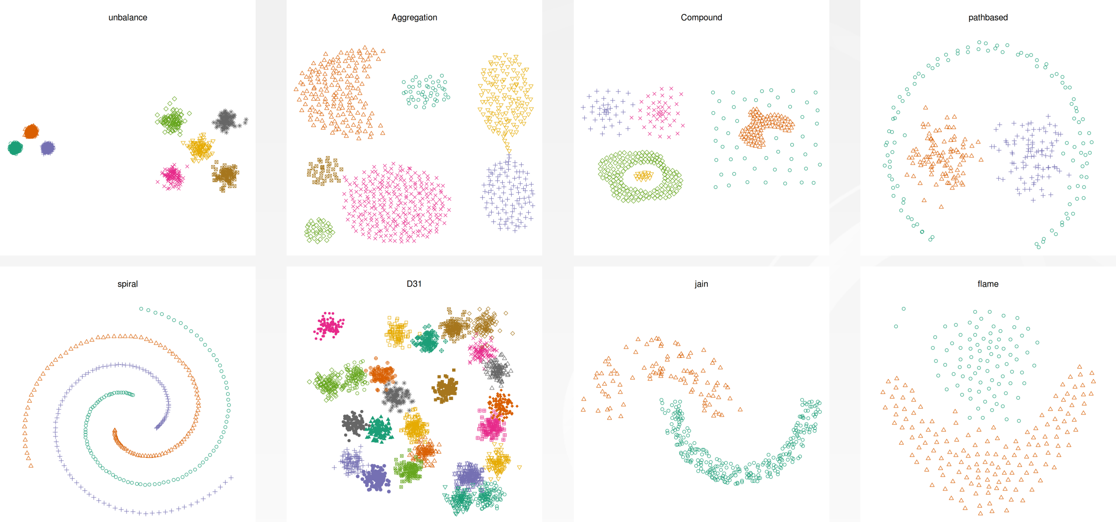
The Genie Algorithm
Benchmark Data

21 Euclidean benchmarks datasets, see gagolewski.com/resources/data/clustering/: Sources:

- A. Gionis et al., Clustering aggregation, ACM Transactions on Knowledge Discovery from Data (TKDD), 2007, pp. 1-30.
- C.T. Zahn, Graph-theoretical methods for detecting and describing gestalt clusters, IEEE Transactions on Computers C-20(1), 1971, pp. 68-86.
- H. Chang, D.Y. Yeung, Robust path-based spectral clustering, Pattern Recognition 41(1), 2008, pp. 191-203.
- C.J. Veenman et al., A maximum variance cluster algorithm, IEEE Transactions on Pattern Analysis and Machine Intelligence 24(9), 2002, pp. 1273-1280.
- A. Jain, M. Law, Data clustering: A user's dilemma, Lecture Notes in Computer Science 3776, 2005, pp. 1-10.
- L. Fu, E. Medico, FLAME, a novel fuzzy clustering method for the analysis of DNA microarray data, BMC bioinformatics 8, 2007, p. 3.
- P. Fränti, O. Virmajoki, Iterative shrinking method for clustering problems, Pattern Recognition, 39(5), 2006, pp. 761-765.
- I. Kärkkäinen, P. Fränti, Dynamic local search algorithm for the clustering problem, Research Report A-2002-6.

Each dataset comes with a sequence of reference labels.

The Genie Algorithm
Benchmark Data



...and many more

The Genie Algorithm

Basic summary statistics of the FM-index distribution over the 21 Euclidean benchmark sets.

	single	complete	ward	average	gini_0.2	gini_0.3	gini_0.4	gini_0.5	gini_0.6	BIRCH	k-means
Min	0.257	0.339	0.337	0.357	0.582	**0.602**	0.563	0.529	0.482	0.350	0.327
Q1	0.480	0.623	0.674	0.707	0.723	0.697	**0.751**	0.742	0.695	0.653	0.701
Median	0.691	0.833	0.842	0.862	**0.923**	0.905	0.828	0.843	0.754	0.894	0.821
Q3	0.764	0.920	0.924	0.936	**0.987**	**0.987**	**0.987**	0.923	0.911	0.924	0.969
Max	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
Mean	0.629	0.777	0.803	0.812	**0.850**	0.841	0.833	0.828	0.789	0.801	0.816
St.Dev.	0.224	0.187	0.177	0.172	0.150	0.146	0.145	**0.138**	0.156	0.183	0.177

Implementation

Implementation

Theorem.
The Genie linkage criterion can be implemented based on an MST.

Implementation

Theorem.
The Genie linkage criterion can be implemented based on an MST.

Two C++ implementations:

Implementation

Theorem.
The Genie linkage criterion can be implemented based on an MST.

Two C++ implementations:

- A modified Prim's algorithm (Olson, 1995):

Implementation

Theorem.
The Genie linkage criterion can be implemented based on an MST.

Two C++ implementations:

- A modified Prim's algorithm (Olson, 1995):
  - paralellizable;

Implementation

Theorem.
The Genie linkage criterion can be implemented based on an MST.

Two C++ implementations:

- A modified Prim's algorithm (Olson, 1995):
  - paralellizable;
  - exactly $(n^2 - n)/2 =: 100\%$ calls to $\mathfrak{d}$;

Implementation

Theorem.
The Genie linkage criterion can be implemented based on an MST.

Two C++ implementations:

- A modified Prim's algorithm (Olson, 1995):
  - paralellizable;
  - exactly $(n^2 - n)/2 =: 100\%$ calls to $\mathfrak{d}$;
- A modified Kruskal's algorithm:

Implementation

Theorem.
The Genie linkage criterion can be implemented based on an MST.

Two C++ implementations:

- A modified Prim's algorithm (Olson, 1995):
  - paralellizable;
  - exactly $(n^2 - n)/2 =: 100\%$ calls to $\mathfrak{d}$;
- A modified Kruskal's algorithm:
  - paralellizable;

Implementation

Theorem.
The Genie linkage criterion can be implemented based on an MST.

Two C++ implementations:

- A modified Prim's algorithm (Olson, 1995):
  - paralellizable;
  - exactly $(n^2 - n)/2 =: 100\%$ calls to $\mathfrak{d}$;
- A modified Kruskal's algorithm:
  - paralellizable;
  - extended vp-trees (Yianilos, 1993) used to speed up seeking NNs;

Implementation

Theorem.
The Genie linkage criterion can be implemented based on an MST.

Two C++ implementations:

- A modified Prim's algorithm (Olson, 1995):
  - paralellizable;
  - exactly $(n^2 - n)/2 =: 100\%$ calls to $\mathfrak{d}$;
- A modified Kruskal's algorithm:
  - paralellizable;
  - extended vp-trees (Yianilos, 1993) used to speed up seeking NNs;
  - between $\sim 1\%$ and (curse of dimensionality) $\sim 120\%$ calls to $\mathfrak{d}$ (empirically).

Implementation

Calls to $\mathfrak{d}$ (relative to $(n^2 - n)/2$); $\mathbb{R}^d$; $\mathcal{N}$(random μs, σ)

σ	d	n	gini_0.3	gini_1.0 (single)	complete	Ward	average
0,50	2	10000	$*$4.8%	†100%	476%	204%	484%
0,50	5	10000	$*$22.0%	†100%	493%	221%	496%
1,50	10	10000	$*$30.3%	†100%	496%	240%	499%
1,50	15	10000	$*$58.3%	†100%	497%	253%	498%
1,50	20	10000	$*$84.9%	†100%	497%	261%	498%
3,50	100	10000	$*$101.8%	†100%	498%	299%	499%
5,00	250	10000	$*$100.9%	†100%	498%	312%	499%

$*$ – modified Kruskal, † – modified Prim

complete, Ward, average – `fastcluster` package for R; see Müllner, 2013; NN-chains etc.

Implementation

Corollary.
There is no need to compute the distance matrix (see `dist()` in R).

Implementation

Corollary.
There is no need to compute the distance matrix (see `dist()` in R).

Example.
$n = 100{,}000 \longrightarrow (n^2 - n)/2 = 4{,}999{,}950{,}000$ floats $\simeq 40$ GB.

See also: `fastcluster::hclust.vector()`

Implementation

A few technical details:

- 2.5k lines of C++ code;
- the Gini-index computed incrementally ($O(n)$ time per iteration);
- OpenMP for multi-threaded computations;
- a custom Union-Find data structure implementation (`disjoint_sets` in Boost are pretty slow);
- a number of tweaks in the VP-tree implementation (search for NNs of x_i only among x_js with $j > i$, make subtrees inactive if all elements are already in the same cluster, etc.).

Implementation

Exemplary run-times for different #threads, $n = 100{,}000$ [s].

			Thread count		
data	Algorithm	g	1	2	4
$d = 10$, $\sigma = 1.5$	Kruskal, VP-trees	0.3	46.5	33.4	28.2
	Prim	0.3	91.5	59.9	44.8
	Kruskal, VP-trees	1.0	32.0	19.1	13.4
	Prim	1.0	77.5	47.7	31.6

Implementation

Exemplary run-times for different #threads, $n = 100{,}000$ [s].

data	Algorithm	g	Thread count 1	2	4
$d = 10, \sigma = 1.5$	Kruskal, VP-trees	0.3	46.5	33.4	28.2
	Prim	0.3	91.5	59.9	44.8
	Kruskal, VP-trees	1.0	32.0	19.1	13.4
	Prim	1.0	77.5	47.7	31.6
	Ward	—	1452.8	—	—

Implementation

Exemplary run-times for different #threads, $n = 100{,}000$ [s].

data	Algorithm	g	Thread count 1	2	4
$d = 10$, $\sigma = 1.5$	Kruskal, VP-trees	0.3	46.5	33.4	28.2
	Prim	0.3	91.5	59.9	44.8
	Kruskal, VP-trees	1.0	32.0	19.1	13.4
	Prim	1.0	77.5	47.7	31.6
	Ward	—	1452.8	—	—
$d = 100$, $\sigma = 3.5$	Kruskal, VP-trees	0.3	1396	740	456
	Prim	0.3	743	413	293
	Kruskal, VP-trees	1.0	1385	717	454
	Prim	1.0	734	396	281

Implementation

Exemplary run-times for different #threads, $n = 100{,}000$ [s].

			Thread count		
data	Algorithm	g	1	2	4
$d = 10$, $\sigma = 1.5$	Kruskal, VP-trees	0.3	46.5	33.4	28.2
	Prim	0.3	91.5	59.9	44.8
	Kruskal, VP-trees	1.0	32.0	19.1	13.4
	Prim	1.0	77.5	47.7	31.6
	Ward	—	1452.8	—	—
$d = 100$, $\sigma = 3.5$	Kruskal, VP-trees	0.3	1396	740	456
	Prim	0.3	743	413	293
	Kruskal, VP-trees	1.0	1385	717	454
	Prim	1.0	734	396	281
	Ward	—	18434	—	—

Ward — `fastcluster::hclust.vector()`

Implementation

- R interface: the `genie` package on CRAN;

Implementation

- R interface: the `genie` package on CRAN;
- Rcpp used to “glue” everything together (see below for discussion);

Implementation

- R interface: the genie package on CRAN;
- Rcpp used to “glue” everything together (see below for discussion);
- Need for speed – R code reduced to:

```
> genie::hclust2
function (d=NULL, objects=NULL, thresholdGini=0.3, useVpTree=FALSE, ...)
{
    opts <- list(thresholdGini=thresholdGini, useVpTree=useVpTree, ...)
    result <- .hclust2_gini(d, objects, opts)  # a call to an Rcpp function
    result[["call"]] <- match.call()
    result[["method"]] <- "gini"
    if (any(result[["height"]] < 0)) {
        # ...
    }
    result
}
```

Implementation

R interface – An example:

```
> h <- genie::hclust2(objects=as.matrix(iris[,1:4]), d="euclidean")
> h
Call:
genie::hclust2(d = "euclidean", objects = as.matrix(iris[, 1:4]))

Cluster method   : gini
Distance         : euclidean
Number of objects: 150
> as.numeric(dendextend::FM_index(cutree(h, 3), iris[[5]]))
[1] 0.9234342
```

Implementation

Inputs:

- `objects` – `NULL` or a numeric matrix or a character vector or a list with integer vectors or . . . ;
- `d` – an object of "class" `dist` or a string: *euclidean, manhattan, minkowski, hamming, levenshtein, dinu, . . .*;
- `thresholdGini` – a single float;
- `useVpTree` – a single bool;

Implementation

Inputs:

- `objects` – NULL or a numeric matrix or a character vector or a list with integer vectors or . . . ;
- `d` – an object of “class“ `dist` or a string: *euclidean, manhattan, minkowski, hamming, levenshtein, dinu, . . .*;
- `thresholdGini` – a single float;
- `useVpTree` – a single bool;

Output:

- An `hclust` “object”, i.e. an R list with attributes:
  - `class`;
  - `names`;

Implementation

Inputs:

- `objects` – NULL or a numeric matrix or a character vector or a list with integer vectors or . . . ;
- `d` – an object of "class" `dist` or a string: *euclidean, manhattan, minkowski, hamming, levenshtein, dinu, . . .*;
- `thresholdGini` – a single float;
- `useVpTree` – a single bool;

Output:

- An `hclust` "object", i.e. an R list with attributes:
  - `class`;
  - `names`;

  and elements:
  - `merge` – matrix with 2 columns and $n-1$ rows;
  - `height` – an integer vector with $n-1$ elements;
  - `order` – an integer vector with n elements;
  - `labels` – a character vector or NULL;

Implementation

We used Rcpp for:

Implementation

We used Rcpp for:

- setting up the package build (register `.Calls`, etc.);

Implementation

We used Rcpp for:

- setting up the package build (register `.Calls`, etc.);
- feeding the algorithms with input data;

Implementation

We used Rcpp for:

- setting up the package build (register `.Calls`, etc.);
- feeding the algorithms with input data;
- constructing the output (named) list easily.

Implementation

We used Rcpp for:

- setting up the package build (register `.Calls`, etc.);
- feeding the algorithms with input data;
- constructing the output (named) list easily.

R and C++ integration is indeed *seamless*.

Implementation

We used Rcpp for:

- setting up the package build (register `.Calls`, etc.);
- feeding the algorithms with input data;
- constructing the output (named) list easily.

R and C++ integration is indeed *seamless*.

We didn't use Rcpp for:

- performing actual computations.

Implementation

We used Rcpp for:

- setting up the package build (register `.Calls`, etc.);
- feeding the algorithms with input data;
- constructing the output (named) list easily.

R and C++ integration is indeed *seamless*.

We didn't use Rcpp for:

- performing actual computations.

Our C++ code is pretty **"R-independent"**.

Summer is Coming (a.k.a. A Possible Generalization)

R is doing its job (and helping us do our job) well today. . .

Summer is Coming (a.k.a. A Possible Generalization)

R is doing its job (and helping us do our job) well today...
...but will it be used in 2020? 2030? ...

Summer is Coming (a.k.a. A Possible Generalization)

R is doing its job (and helping us do our job) well today...
...but will it be used in 2020? 2030? ...

We felt that limiting our “audience” only to R users would be a seriously wrong idea.

Summer is Coming (a.k.a. A Possible Generalization)

R is doing its job (and helping us do our job) well today...
...but will it be used in 2020? 2030? ...

We felt that limiting our “audience” only to R users would be a seriously wrong idea.

Thanks to the C++ code design we are able to effortlessly deliver interfaces for:

Summer is Coming (a.k.a. A Possible Generalization)

R is doing its job (and helping us do our job) well today...
...but will it be used in 2020? 2030? ...

We felt that limiting our “audience” only to R users would be a seriously wrong idea.

Thanks to the C++ code design we are able to effortlessly deliver interfaces for:

- Python;

Summer is Coming (a.k.a. A Possible Generalization)

R is doing its job (and helping us do our job) well today...
...but will it be used in 2020? 2030? ...

We felt that limiting our "audience" only to R users would be a seriously wrong idea.

Thanks to the C++ code design we are able to effortlessly deliver interfaces for:

- Python;
- Julia;

Summer is Coming (a.k.a. A Possible Generalization)

R is doing its job (and helping us do our job) well today...
...but will it be used in 2020? 2030? ...

We felt that limiting our "audience" only to R users would be a seriously wrong idea.

Thanks to the C++ code design we are able to effortlessly deliver interfaces for:

- Python;
- Julia;
- Matlab;

Summer is Coming (a.k.a. A Possible Generalization)

R is doing its job (and helping us do our job) well today...
...but will it be used in 2020? 2030? ...

We felt that limiting our “audience” only to R users would be a seriously wrong idea.

Thanks to the C++ code design we are able to effortlessly deliver interfaces for:

- Python;
- Julia;
- Matlab;
- etc. (there are plenty of fish in the sea...)

Summer is Coming (a.k.a. A Possible Generalization)

Advantages of “portable” (end-user language-agnostic) code:

Summer is Coming (a.k.a. A Possible Generalization)

Advantages of “portable” (end-user language-agnostic) code:

- Write once, let others do more.

Summer is Coming (a.k.a. A Possible Generalization)

Advantages of "portable" (end-user language-agnostic) code:

- Write once, let others do more.
- Well-tested, reliable, efficient.

Summer is Coming (a.k.a. A Possible Generalization)

Advantages of “portable” (end-user language-agnostic) code:

- Write once, let others do more.
- Well-tested, reliable, efficient.
- R+`stats`+...and Python+`numpy`+`sklearn`+...are pretty similar.

Summer is Coming (a.k.a. A Possible Generalization)

Advantages of “portable” (end-user language-agnostic) code:

- Write once, let others do more.
- Well-tested, reliable, efficient.
- R+`stats`+…and Python+`numpy`+`sklearn`+…are pretty similar.
- Use cases: numerical/scientific computing, simulations, machine learning…

Summer is Coming (a.k.a. A Possible Generalization)

Advantages of “portable” (end-user language-agnostic) code:

- Write once, let others do more.
- Well-tested, reliable, efficient.
- R+`stats`+…and Python+`numpy`+`sklearn`+…are pretty similar.
- Use cases: numerical/scientific computing, simulations, machine learning…
- Core algorithms are implemented in C/C++/Fortran anyway.

Summer is Coming (a.k.a. A Possible Generalization)

Advantages of “portable” (end-user language-agnostic) code:

- Write once, let others do more.
- Well-tested, reliable, efficient.
- R+`stats`+...and Python+`numpy`+`sklearn`+...are pretty similar.
- Use cases: numerical/scientific computing, simulations, machine learning...
- Core algorithms are implemented in C/C++/Fortran anyway.
- Python has it, R doesn’t (or vice versa) – no more an issue.

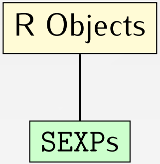
Summer is Coming (a.k.a. A Possible Generalization)

The Rcpp way:

R Objects

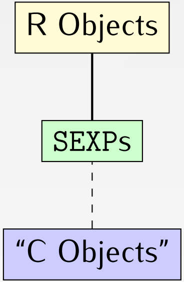
Summer is Coming (a.k.a. A Possible Generalization)

The Rcpp way:



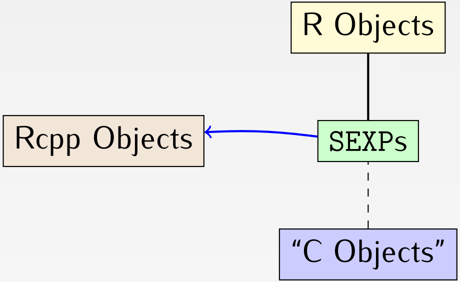
Summer is Coming (a.k.a. A Possible Generalization)

The Rcpp way:



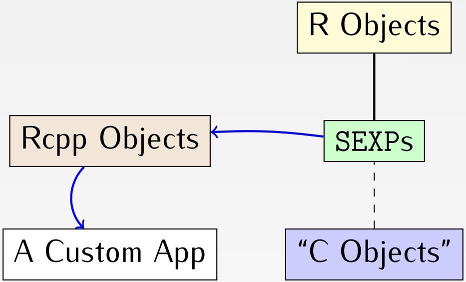
Summer is Coming (a.k.a. A Possible Generalization)

The Rcpp way:



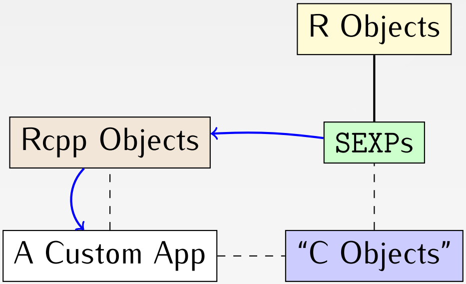
Summer is Coming (a.k.a. A Possible Generalization)

The Rcpp way:



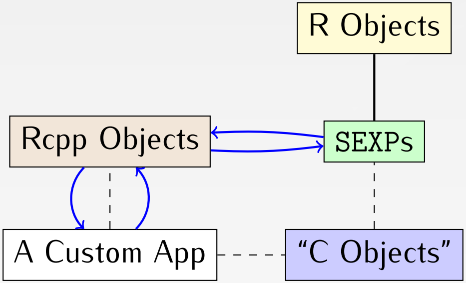
Summer is Coming (a.k.a. A Possible Generalization)

The Rcpp way:



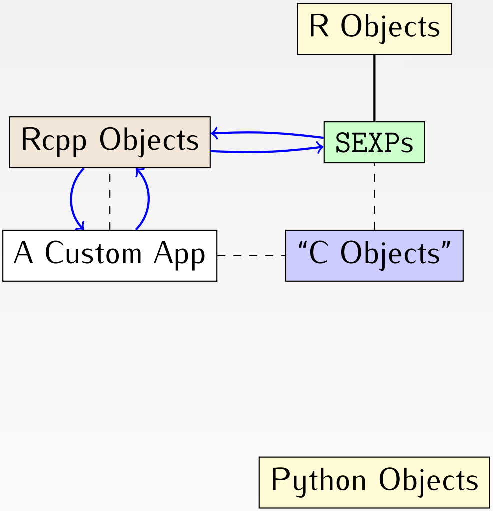
Summer is Coming (a.k.a. A Possible Generalization)

The Rcpp way:



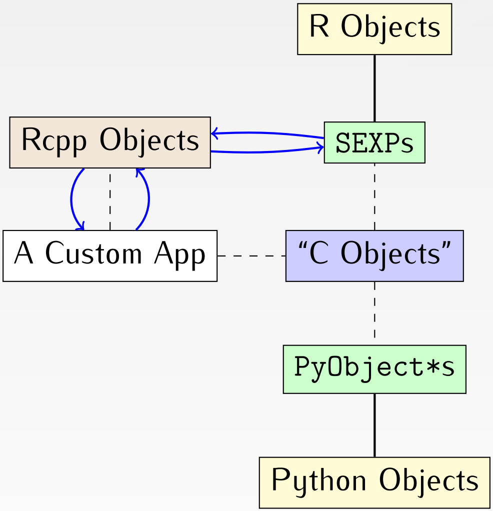
Summer is Coming (a.k.a. A Possible Generalization)

The Rcpp way:



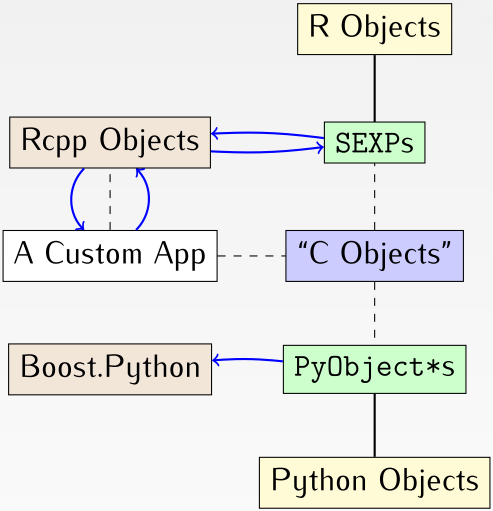
Summer is Coming (a.k.a. A Possible Generalization)

The Rcpp way:



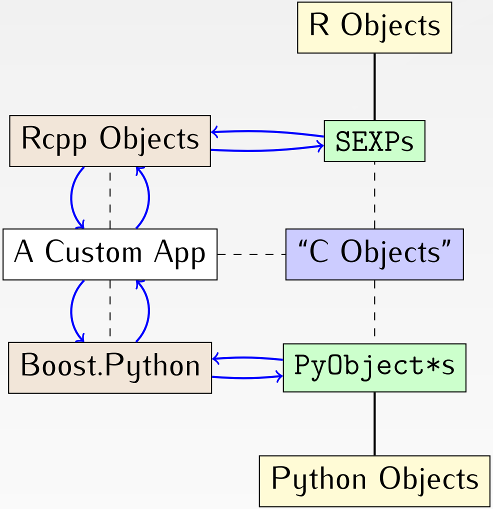
Summer is Coming (a.k.a. A Possible Generalization)

The Rcpp way:



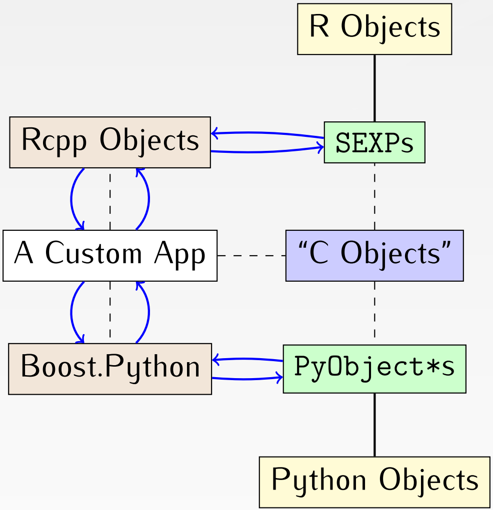
Summer is Coming (a.k.a. A Possible Generalization)

The Rcpp way:

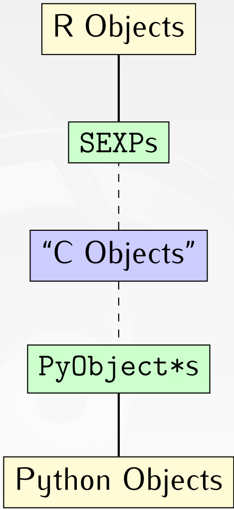


Summer is Coming (a.k.a. A Possible Generalization)

The Rcpp way:

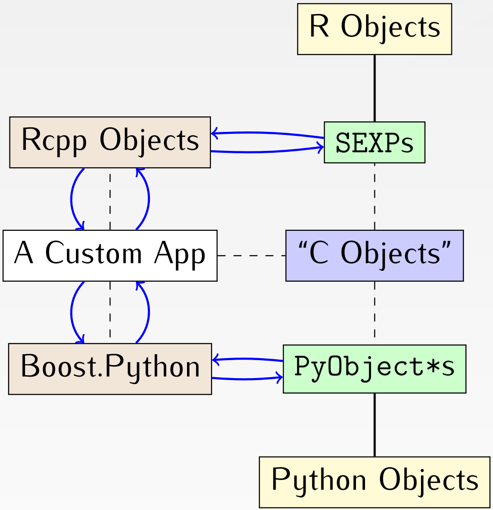


New project (contributors needed!):

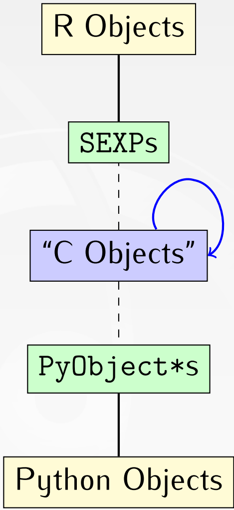


Summer is Coming (a.k.a. A Possible Generalization)

The Rcpp way:

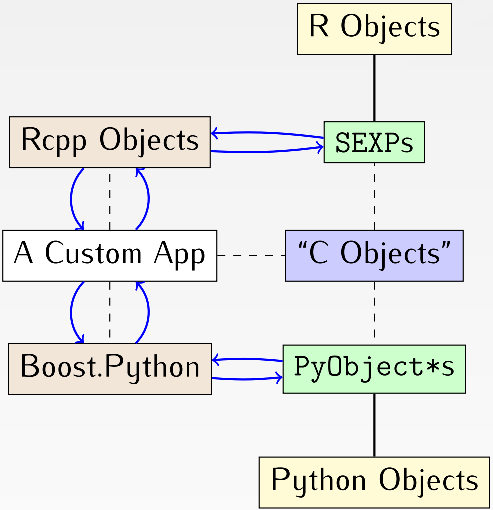


New project (contributors needed!):

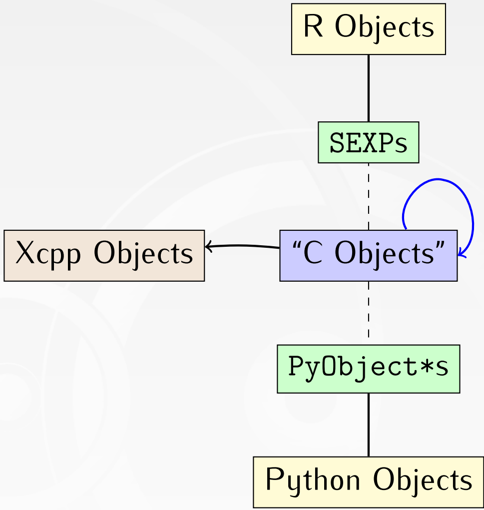


Summer is Coming (a.k.a. A Possible Generalization)

The Rcpp way:



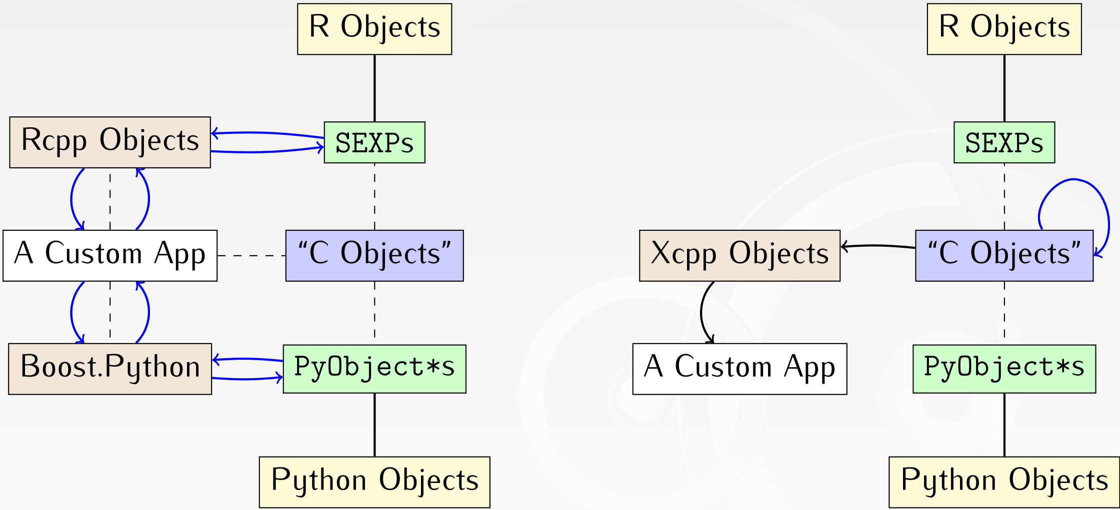
New project (contributors needed!):



Summer is Coming (a.k.a. A Possible Generalization)

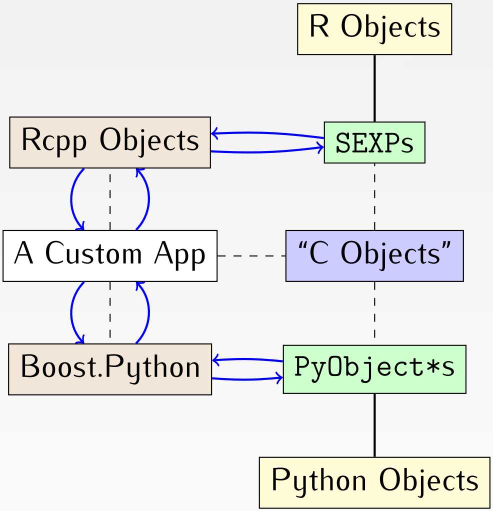
The Rcpp way:

New project (contributors needed!):

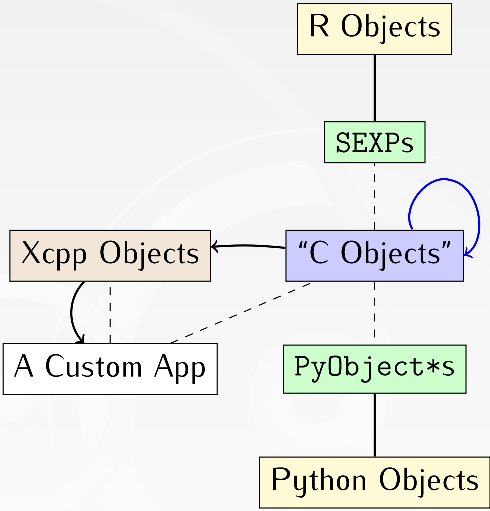


Summer is Coming (a.k.a. A Possible Generalization)

The Rcpp way:

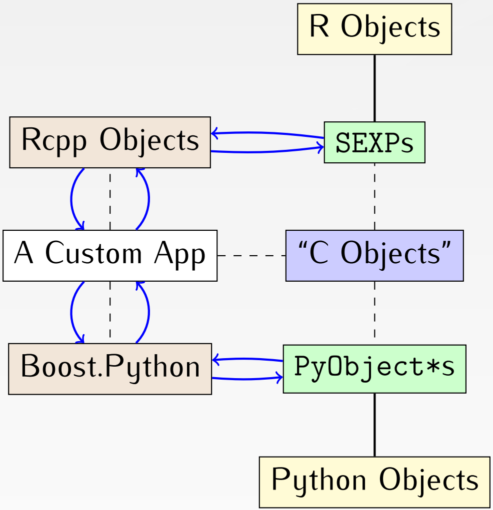


New project (contributors needed!):

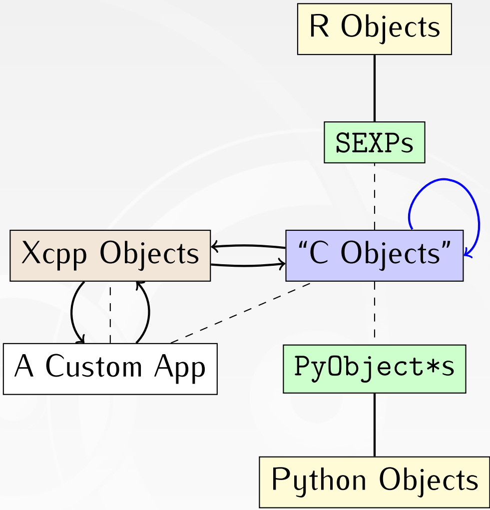


Summer is Coming (a.k.a. A Possible Generalization)

The Rcpp way:

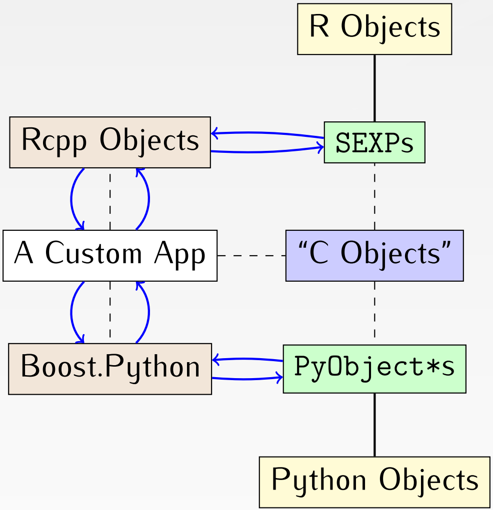


New project (contributors needed!):

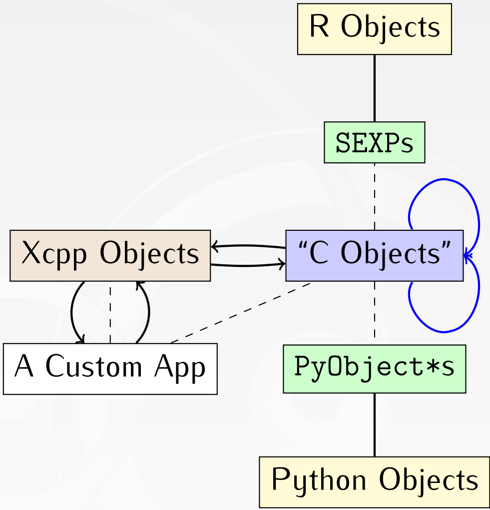


Summer is Coming (a.k.a. A Possible Generalization)

The Rcpp way:



New project (contributors needed!):



Summer is Coming (a.k.a. A Possible Generalization)

New project (contributors needed!):

- Xcpp: Language-agnostic C++ bindings for scientific computing;

Summer is Coming (a.k.a. A Possible Generalization)

New project (contributors needed!):

- Xcpp: Language-agnostic C++ bindings for scientific computing;
- Common, most useful objects:

Summer is Coming (a.k.a. A Possible Generalization)

New project (contributors needed!):

- Xcpp: Language-agnostic C++ bindings for scientific computing;
- Common, most useful objects:
  - bool, int, float, complex, string scalars

Summer is Coming (a.k.a. A Possible Generalization)

New project (contributors needed!):

- Xcpp: Language-agnostic C++ bindings for scientific computing;
- Common, most useful objects:
  - bool, int, float, complex, string scalars
  - bool, int, float, complex, string vectors;

Summer is Coming (a.k.a. A Possible Generalization)

New project (contributors needed!):

- Xcpp: Language-agnostic C++ bindings for scientific computing;
- Common, most useful objects:
  - bool, int, float, complex, string scalars
  - bool, int, float, complex, string vectors;
  - bool, int, float, complex, string matrices
    ("long" vectors with 2D indexing, C- or Fortran-order);

Summer is Coming (a.k.a. A Possible Generalization)

New project (contributors needed!):

- Xcpp: Language-agnostic C++ bindings for scientific computing;
- Common, most useful objects:
  - bool, int, float, complex, string scalars
  - bool, int, float, complex, string vectors;
  - bool, int, float, complex, string matrices
    ("long" vectors with 2D indexing, C- or Fortran-order);
  - "lists" (sequences of arbitrarily-typed elements, integer-based indexing);

Summer is Coming (a.k.a. A Possible Generalization)

New project (contributors needed!):

- Xcpp: Language-agnostic C++ bindings for scientific computing;
- Common, most useful objects:
  - bool, int, float, complex, string scalars
  - bool, int, float, complex, string vectors;
  - bool, int, float, complex, string matrices
    (“long” vectors with 2D indexing, C- or Fortran-order);
  - “lists” (sequences of arbitrarily-typed elements, integer-based indexing);
  - “dictionaries” (key-value pairs, string-based indexing)
    (in R – named lists, “objects”, environments);

Summer is Coming (a.k.a. A Possible Generalization)

New project (contributors needed!):

- Xcpp: Language-agnostic C++ bindings for scientific computing;
- Common, most useful objects:
  - bool, int, float, complex, string scalars
  - bool, int, float, complex, string vectors;
  - bool, int, float, complex, string matrices
    (“long” vectors with 2D indexing, C- or Fortran-order);
  - “lists” (sequences of arbitrarily-typed elements, integer-based indexing);
  - “dictionaries” (key-value pairs, string-based indexing)
    (in R – named lists, “objects”, environments);
  - “XPtrs” for auxiliary, “internal” data to be alive between function calls;

Summer is Coming (a.k.a. A Possible Generalization)

New project (contributors needed!):

- Xcpp: Language-agnostic C++ bindings for scientific computing;
- Common, most useful objects:
  - bool, int, float, complex, string scalars
  - bool, int, float, complex, string vectors;
  - bool, int, float, complex, string matrices
    ("long" vectors with 2D indexing, C- or Fortran-order);
  - "lists" (sequences of arbitrarily-typed elements, integer-based indexing);
  - "dictionaries" (key-value pairs, string-based indexing)
    (in R – named lists, "objects", environments);
  - "XPtrs" for auxiliary, "internal" data to be alive between function calls;
- Note that R/Pandas data frames are somewhere in-between matrices and dictionaries;

Summer is Coming (a.k.a. A Possible Generalization)

New project (contributors needed!):

- Xcpp: Language-agnostic C++ bindings for scientific computing;
- Common, most useful objects:
  - bool, int, float, complex, string scalars
  - bool, int, float, complex, string vectors;
  - bool, int, float, complex, string matrices
    ("long" vectors with 2D indexing, C- or Fortran-order);
  - "lists" (sequences of arbitrarily-typed elements, integer-based indexing);
  - "dictionaries" (key-value pairs, string-based indexing)
    (in R – named lists, "objects", environments);
  - "XPtrs" for auxiliary, "internal" data to be alive between function calls;
- Note that R/Pandas data frames are somewhere in-between matrices and dictionaries;
- Of course, Rcpp-like "sugar" ops can be added (overloaded operators, etc.).

Summer is Coming (a.k.a. A Possible Generalization)

New project (contributors needed!) – Some issues :

Summer is Coming (a.k.a. A Possible Generalization)

New project (contributors needed!) – Some issues :

- input/output data preparation – some additional time & RAM needed;

Summer is Coming (a.k.a. A Possible Generalization)

New project (contributors needed!) – Some issues :

- input/output data preparation – some additional time & RAM needed;
- sometimes in/out objects must be pre/post-processed in R/Python;

Summer is Coming (a.k.a. A Possible Generalization)

New project (contributors needed!) – Some issues :

- input/output data preparation – some additional time & RAM needed;
- sometimes in/out objects must be pre/post-processed in R/Python;
- not every “data type” will be covered – can’t be language-dependent;

Summer is Coming (a.k.a. A Possible Generalization)

New project (contributors needed!) – Some issues :

- input/output data preparation – some additional time & RAM needed;
- sometimes in/out objects must be pre/post-processed in R/Python;
- not every "data type" will be covered – can't be language-dependent;
- serialization of auxiliary objects – lack of a proper R interface;

Summer is Coming (a.k.a. A Possible Generalization)

New project (contributors needed!) – Some issues :

- input/output data preparation – some additional time & RAM needed;
- sometimes in/out objects must be pre/post-processed in R/Python;
- not every "data type" will be covered – can't be language-dependent;
- serialization of auxiliary objects – lack of a proper R interface;
- but seamless mappings between C++ and standard Python or S4/RefClasses/R6 objects may be introduced;

Summer is Coming (a.k.a. A Possible Generalization)

New project (contributors needed!) – Some issues :

- input/output data preparation – some additional time & RAM needed;
- sometimes in/out objects must be pre/post-processed in R/Python;
- not every “data type” will be covered – can’t be language-dependent;
- serialization of auxiliary objects – lack of a proper R interface;
- but seamless mappings between C++ and standard Python or S4/RefClasses/R6 objects may be introduced;
- accessing/syncing RNG state;

Summer is Coming (a.k.a. A Possible Generalization)

New project (contributors needed!) – Some issues :

- input/output data preparation – some additional time & RAM needed;
- sometimes in/out objects must be pre/post-processed in R/Python;
- not every "data type" will be covered – can't be language-dependent;
- serialization of auxiliary objects – lack of a proper R interface;
- but seamless mappings between C++ and standard Python or S4/RefClasses/R6 objects may be introduced;
- accessing/syncing RNG state;
- build & link – two possibilities:
  - independent builds for each language – not so efficient (time and space);
  - one "core" dynamically linked library, with separate language-specific interfaces that link to it – relying on C++-based libs may be hard (`extern "C"`, different compilers, etc. – e.g., R-tools gcc vs Anaconda MSVC).
- ...

Thank you!

O. Aristondo, J. García-Lapresta, C. Lasso de la Vega, and R. Marques Pereira. Classical inequality indices, welfare and illfare functions, and the dual decomposition. *Fuzzy Sets and Systems*, 228:114–136, 2013.

G. Beliakov, M. Gagolewski, and S. James. Penalty-based and other representations of economic inequality. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 2016. in press.

M. M. Deza and E. Deza. *Encyclopedia of Distances*. Springer, 2014.

M. Gagolewski, M. Bartoszuk, and A. Cena. Genie: A new, fast, and outlier-resistant hierarchical clustering algorithm. *Information Sciences*, 363:8–23, 2016a.

M. Gagolewski, A. Cena, and M. Bartoszuk. Hierarchical clustering via penalty-based aggregation and the Genie approach. In V. Torra et al., editors, *Modeling Decisions for Artificial Intelligence*, volume 9880 of *Lecture Notes in Artificial Intelligence*, pages 191–202. Springer, 2016b.

D. Müllner. fastcluster: Fast hierarchical, agglomerative clustering routines for R and Python. *Journal of Statistical Software*, 53(9):1–18, 2013.

C. F. Olson. Parallel algorithms for hierarchical clustering. *Parallel Computing*, 21: 1313–1325, 1995.

P. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proc. ACM-SIAM Symp. Discrete Algorithms*, pages 311–321. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1993.