Google Summer Of Code 2017
A Unified R Package for
Estimating the Empirical Cluster Tree

Matthew Piekenbrock

April 2, 2017

Abstract

This document is a student application for a project proposed under the R language and environment for statistical computing and graphics organization titled Estimating the Empirical Cluster Tree

Project Info

Project Title: Estimating the Empirical Cluster Tree
Project short title: A 'clustertree' package for R
Project listing:
https://github.com/rstats-gsoc/gsoc2017/wiki/Estimating-the-empirical-Cluster-Tree

Name and Contact Information

Student name: Matthew Piekenbrock
Student postal address: 2399 Sydneys Bend Drive, Miamisburg, OH 45342
Telephone(s): 937-269-8582
Email(s): matt.piekenbrock@gmail; piekenbrock.5@wright.edu
Other communications channels: http://mattpiekenbrock.com

Student Affiliation

Institution: Wright State University
Program: Masters of Science in Computer Science
Stage of completion: Beginning of 2nd year
Contact to verify: Assistant Professor Derek Doran (derek.doran@wright.edu)

Mentors

Mentor names: Mikhail Belkin, Michael Hahsler
Mentor emails: mbelkin@cse.ohio-state.edu, mhahsler@lyle.smu.edu
Other communications channels:
http://web.cse.ohio-state.edu/~mbelkin/, http://michael.hahsler.net/

Student Bio

Experience

I am a graduate research assistant currently pursuing my Masters of Science in Computer Science at Wright State University, with the intent of pursuing a doctorate of Computer Science starting Fall of 2018. Beginning Fall 2013, I worked in a research position for Dr. Andrew Terzuoli at the Air Force Institute of Technology (AFIT) as an undergraduate research assistant. I worked at AFIT until I graduated with my Bachelors of Science in Computer Science in the Fall of 2015, where I continued research work with reduced hours until Spring 2017. I also worked as an undergraduate research assistant for Dr. Derek Doran in the Web and Complex Systems lab (WaCS) starting in Fall 2017 doing research that I then continued as graduate research assistant starting in the Spring of 2016. I currently maintain this position during the regular academic year.

Background

In my undergraduate years at AFIT, I worked in an educational research group with a number of undergraduate and graduate students on a diverse range of problems within the realm of scientific computing. Many of these projects are closed-source, however one such research effort relevant to the proposed project has lead to two publications (see [29] and [28]), which involved a novel technique for doing Iterative Closest Point (ICP) using CUDA. Briefly, one of the bottlenecks dominating the computation of the ICP algorithm is a variant of the standard k-nearest neighbor (kNN) problem, where (in the ICP context) the goal is to calculate, for every point from a source data set Q, its $k = 1$ nearest neighbor in a reference data set R. The project involved improving standard approaches to ICP. The solution we proposed requires augmenting the traditional data structures used in various kNN applications in such a way that exploits the intrinsic parallelism of the problem while minimizing suboptimal memory access issues that are well-known to plague uncoalesced memory transactions. During this research period, I studied many modern data structures and approaches to the kNN problem (including kd Trees, cover trees, locality sensitive hashing methods, etc.). I also became fairly proficient with a few parallel architectures (CUDA, OpenCL), modern C++ optimization, and generic programming techniques.

When I started as graduate research assistant for the WaCS lab, I assisted with a research project that dealt with the unsupervised problem of discovering meaningful locations within spatial data sets observed over large geographical areas. My current research has continued on this project, which involves improving the capability of various clustering models to intrinsically identify "locations" within unlabeled surveillance data, validated against existing knowledge bases (such as Google Places, OpenStreetMap, private sources of 'truth', etc.). During the preliminary literature review, it was discovered that the location discovery and recommendation field is largely saturated with density-based clustering and/or network-based approaches, which lead me to delve into the internals of such methods. In particular, I discovered that the density-based clustering field offered extremely scalable and useful techniques for the geospatial domain. Despite the utility of such methods, the statistical properties of most density-based algorithms are generally unknown, preventing the possibility of doing statistical inference.

In pursuit of an approach that allows for formal analysis yet maintains the strengths of practical density-based algorithms, I became engrossed by the underlying statistical theory of clustering. Perhaps the most popular algorithm in the applied density-based clustering field—having won SIGKDD 2014's Test of Time Award [31]—is the DBSCAN algorithm [11]. Although algorithmically simple and intuitive to understand, I noticed DBSCAN manifesting in many applied contexts, which lead me to attempt to understand *why* DBSCAN has proven so useful. This trail of research also lead me to contribute a number of additions to the popular **dbscan** R package. A few authors have recently noticed many commonalities between DSBCAN and more statistically-principled concepts [4] [33]. It was this research that lead me to the theory underlying the 'cluster tree' and the associated notion of fractional consistency put forth by John Hartigan in his seminal paper on the consistency of single linkage [14]. Although the theory by itself is appealing in the sense that it offers a reasonable definition of what a 'natural cluster' is, the direct use of recent, significant advances to the cluster tree theory to solve applied problems seems virtually nonexistent (by the author's knowledge). This application is a brief summary of such advances, why I think they are relevant and novel, and why a common R package that brings the tools used in applied density-based clustering together with the theoretical advances of the cluster tree is important to the progression of the clustering field as a whole. Furthermore, my unique skill set, verifiable experience, and formal statistical background make me an excellent candidate for this project.

Coding Plan and Methods

Project overview

The purpose of this project is to develop an efficient C++/**Rcpp** implementation of the first provably consistent algorithm for estimating the cluster tree, referred to as Robust Single Linkage (RSL) algorithm [6]. Additionally, a secondary goal is to modularize the algorithmic scheme of RSL and extensions to the RSL algorithms (referred to here as RSL variants, explained more below) used to estimate the cluster tree. The outcome of such a package would not only unify recent extensions to the cluster tree into a simple R package, but also it would bring the theory of empirical approximations of the cluster tree to a state where they are *easily usable for applications* that seek to perform formal hierarchical cluster analysis on sufficiently large data sets.

Package design

The design focus of the package is centered on usability, scalability, and extensibility. This implies that the algorithms related to the cluster tree and its related extensions will, when appropriate, be based around standard data structures and classes often used in clustering and familiar to most R users. For example, the representation of the "cluster tree" is a hierarchical representation of connected components; so it would be most useful to represent the tree as a valid `hclust` or `dendrogram` object, with possible conversion methods to graph-based representations (e.g., adjacency list, matrix, and edgelists). Since a primary focus of the proposed package is scalability, any of the computationally difficult tasks would be written in Rcpp/C++. To illustrate extensibility, algorithms to compute various statistics between multiple cluster tree approximations (such as merge distortion [10]) will be included. The intent is that this package can serve as a unified implementation to incorporate ongoing advancements and extensions to the cluster tree itself. The package will also include useful visualizations (such as contour plots, specializations to existing `plot.dendrogram` S3 method, etc.), with the intention of simplifying analysis of the resulting tree and its related algorithms.

Background

Consider the context and general goal of clustering and its relationship to statistical learning. A foremost goal of statistical learning is to understand what a finite data set reveals about the underlying distribution from which the data were sampled [6], whereas the goal of clustering is to group a given collection of unlabeled patterns into *meaningful* clusters [19]. The effectiveness, evaluation strategies, and limitations of *parametric* clustering models—such as the popular Gaussian Mixtures Models (GMMs) and K-means models—are well studied. In contrast to such model-based approaches, a widely used class of algorithms that fall under the category of *hierarchical clustering* have been developed for the purpose of representing data in terms of a certain tree structure [18] [10], however, the statistical properties of many such clustering algorithms are generally unknown, prohibiting the capability of performing formal inference [12]. From the statistical learning perspective, a (highly) desirable goal of hierarchical clustering is understand what the nested, hierarchical relationships obtained from a finite data sample reveal about the true relationships of the underlying population from which the data were observed. This goal shares many commonalities with the field *density-based* clustering, which comprises a set of algorithms and techniques that define clusters to be "contiguous region[s] of high point density, separated from other such clusters by contiguous regions of low point density" [30], following some predefined notion of density. But even given an appropriate definition of density, how does one go about selecting these 'regions' of low densities that delimit disjoint clusters? Intuitively, an alternative definition of density-based clustering is one that identifies clusters based on *the order relationships the data reveals about the underlying population density.* Viewed from the latter definition, the field of hierarchical and density-based clustering have remarkably similar analysis goals, and as such, density-based clustering methods stand as a popular paradigm for incorporating elements from *nonparametric* statistics [5]. From the applied perspective, density-based clustering methods also exhibit many favorable characteristics; they are generally capable of finding clusters of 'arbitrary' shape, they are usually robust to noise, they often capture clusters with minimal *a priori* information (i.e., it is often not required to specify the number of clusters, k), and they are often much more computationally efficient relative to model-based approaches [30] [5] [23] [11].

In this application, I follow some previous work on clustering [15]. For a density f on $\mathbb{R}^d$, a *high-density cluster* is any connected component $\mathbb{C}_f(\lambda)$ of $\{x : f(x) \geq \lambda\}$, for some $\lambda > 0$. The set of high-density

clusters across all possible values of λ forms an (infinite) hierarchy called the *cluster tree* of f. Following the notation above, one of the general tasks of hierarchical clustering is to, given a sample $X_n \subset \mathbb{X}$ of size n, construct an *empirical* tree $\hat{\mathbb{C}}_{f_n}$ that estimates the *cluster tree* $\mathbb{C}_f$ of f. A natural goal of such a class of algorithms would be to, as $n \to \infty$, establish a notion of *consistency* in the estimation process[1]. John Hartigan gives a reasonable definition of consistency [14][2]:

Definition 1. *For any sets $A, A' \subset \mathbb{X}$, let A_n (respectively, A'_n) denote the smallest cluster of $\hat{\mathbb{C}}_{f_n}$ containing $A \cap X_n$ (respectively $A' \cap X_n$). We say $\hat{\mathbb{C}}_{f_n}$ is consistent if, whenever A and A' are different connected components of $\{x : f(x) \geq \lambda\}$ (for some $\lambda > 0$), $\mathbb{P}(A_n$ is disjoint from $A'_n) \to 1$ as $n \to \infty$*

In 1981, Hartigan established that the single linkage algorithm is a consistent estimator of the cluster tree for densities in $\mathbb{R}$ [14] (for $d = 1$) however *inconsistent* for any $d \geq 2$. The problem is actually a very intuitive result of the requirement that $A \cap X_n \subset A_n$: consider the situation where a clustering algorithm attempts to capture all of A—there is a reasonable chance that the resulting cluster A_n, containing a portion of A, also contains part of A'. As a result, if $d \geq 2$, any single-linkage cluster containing all of the sample points of A will also contain nearly all of the sample points of A', in probability as $n \to \infty$. See Theorem 1 of [14] for the proof of this.

From this concept, Hartigan deviates from the traditional notion of consistency by defining *fractional consistency* (also sometimes called *Hartigan consistency*), founded on the idea that A_n (respectively, A'_n) need not contain all of $A \cap X_n$ (respectively, $A' \cap X_n$), but it should contain a large portion of it—and that portion should be arbitrarily close (A_n contains a positive fraction of points in A, passing arbitrarily close to every point in A, at distance $\to 0$ as $n \to \infty$). As above, see any of [14] [22] [10] [6] [7] for more details.

Computing a cluster tree: current practice and limitations

Under this notion of consistency, an ideal approach to 'density-based' clustering might involve the study and use of uniformly consistent kernel density estimators (KDE), i.e. the use of approaches that estimate the density of a sample, given some $x \in X_n$:

$$\hat{p} = \frac{1}{n} \sum_{i=1}^{N} \kappa_h(x - x_i)$$

where κ is a 'smoothing' kernel that satisfies a few convenient properties[3], and the subscript h is the well-known bandwidth parameter. After the model is 'fitted' with the appropriate kernel and bandwidth parameter, the data sample can act as input to the resulting estimated density through a post-processing algorithm as a means of producing high-density clusters. The difficulty is that $\hat{\mathbb{C}}_{f_n}$ is **not easy to compute** for typical density estimates f_n (see the appendix, section 5 in [6] for details). As a result, attempts to approximate $\hat{\mathbb{C}}_{f_n}$ have been made in several cases, usually using either parametric or nearest neighbor estimates, but generally without proof of convergence [36] [35]. Furthermore, from a practical standpoint, such models generally require a significant amount of memory to store, and are often computationally inefficient for large data sets [26].

Robust Single Linkage: In 2010, the first provably consistent algorithm for estimating the cluster tree for densities in $\mathbb{R}^d$, referred to (as above) as the Robust Single Linkage (RSL) algorithm, was published [6]. It is not only a tractable algorithm, but it also maintains much of the elegance an simplicity of single-linkage. This is significant to the clustering field: insofar as statistical learning is concerned about what a finite data sample reveals about the underlying density f, density-based clustering methods are concerned with finding clusters based on *meaningful* notions on density. This is, in essence, what many applied density-based clustering algorithms attempt to approximate [17] [8] [20]. Campello *et al* go so far as to assert that nearly all density-based clustering algorithms are, either strictly or loosely, based on Hartigan's model [5].

Consider the RSL algorithm posed by Kamalika Chaudhuri and Sanjoy Dasgupta (henceforth referred to as KC and SD) for estimating the cluster tree from a finite data sample $X_n = \{x_1, \ldots, x_n\}$ sampled from a density f on $\mathbb{X}$, where $\mathbb{X}$ is a subset $\mathbb{R}^d$. Let the distances denoted with $\|\cdot\|$ be exclusively Euclidean

[1] Recall that an estimator $\hat{\Theta}_n$ of a parameter θ if, as more samples are collected ($n \to \infty$), the estimator converges in probability to the true value of the parameter, $\operatorname{plim}_{n\to\infty}(\hat{\Theta}_n) = \theta$

[2] To make things more explicit, I deviate slightly from reference notation and use $\hat{\mathbb{C}}_{f_n}$ to denote the *empirical* cluster tree. This is to emphasize that the set of connected components ($\hat{\mathbb{C}}$), created from a finite sample of size n, are estimates of the true cluster tree ($\mathbb{C}_f$) for a population density f on $\mathbb{R}^d$.

[3] See section 14.7. of [26]

distances on $\mathbb{X}$, and let $B(x, r)$ be the closed ball of radius r around x. The algorithm given in [6] is as follows:

1. For each x_i set $r_k(x_i) = \inf\{r : B(x_i, r) \text{ contains } k \text{ data points}\}$.
2. As r grows from 0 to ∞:
   (a) Construct a graph G_r with nodes $\{x_i : r_k(x_i) \leq r\}$.
   Include edge (x_i, x_j) if $\|x_i - x_j\| \leq \alpha r$
   (b) Let $\hat{\mathbb{C}}(r)$ be the connected components of G_r.

One of the practical difficulties with implementing this algorithm is in the dermination of the critical radii r that connect the density upper level sets, represented using graph terminology as the connected components (CCs) $\hat{\mathbb{C}}(r)$[4] of G_r. To determine such radii requires knowledge of which $x_i \in X_n$ are connected at every level of G_r, implying all $\binom{n}{2}$ pairwise distances need to be computed. Additional overhead exists in efficiently tracking the memberships of each x_i at each level, $\hat{\mathbb{C}}(r)$. An example of this simple algorithm for calculating the connected components following Wishart's original scheme [35] might algorithmically be stated as follows:

Given a finite sample X_n, compute and store the radius to the k^{th} nearest neighbor for each $x_i \in X_n$, denoted as $r_k(x_i)$, such that $r_k(x_i) = \inf\{r : B(x_i, r) \text{ contains } k \text{ data points}\}$. Compute and sort the $\binom{n}{2}$ Euclidean distances in increasing order, denoted as R. Iterate through these distances $r \in R$, connecting the 'edges' (x_i, x_j) subject to the constraint that $\|x_i - x_j\| \leq r_k(x_i)$ and $\|x_i - x_j\| \leq r_k(x_j)$. Track the connected components of the G_r, recording the smallest radius r that changes component membership. The resulting components build an approximation of the hierarchy $\mathbb{C}_{f_n}$, what many are refer to as the cluster tree.

Below is a direct, naïve R implementation of this algorithm using the **igraph** and **FNN** packages, with parameters $k = 5$ and $\alpha = 1$[5].:

R Example 0.1.

```r
suppressMessages({ require("igraph"); require("FNN") })
data(iris)
X <- iris[1:50, 1:4]

## Initialize variables
{ n <- nrow(X); dist_iris <- dist(X); k <- 5 }

## Get the smallest radius as a starting point for r
r <- min(dist_iris)

## Start off with empty adjacency graph
G_r <- igraph::graph_from_adjacency_matrix(diag(n))

## Get balls centered at each x of radius r_k
## (containing k points inclusive of x_i itself)
r_k <- apply(FNN::knn.dist(X, k = k - 1, algorithm = "kd_tree"), 1, max)

## Initiate cluster tree and distance matrix to simplify indexing
clustertree <- list()
l2_dist <- as.matrix(dist_iris)
diag(l2_dist) <- Inf

## Vector of sorted radii to iterate through and a counter
lambda <- sort(dist_iris)
alpha <- 1
i <- 1
```

[4]Note that KC and SD use $\hat{\mathbb{C}}(r)$ to emphasize r as an algorithmic component. The result of $\hat{\mathbb{C}}(r)$ effectively represents $\mathbb{C}_{f_n}$ following the notation from the background section, and thus they will be used interchangeably in this paper.

[5]It's worth noting that consistency was shown for $\alpha \geq \sqrt{2}$.

```
## expand eps-Ball from 0 -> Inf
for (r in lambda){

  ## Wisharts scheme: Only connect points that have at least
  ## k neighbors within distance r
  eps <- mapply(function(i, j) ( l2_dist[i, j] <= r * alpha &&
                                 r_k[i] <= r && # radius of x_i
                                 r_k[j] <= r ), # radius of x_j
                row(l2_dist), col(l2_dist))

  ## Construct the adjacency graph, recording distinct CCs as the level sets
  adj_matrix <- matrix(as.integer(eps), nrow = n, ncol = n)
  G_r <- igraph::graph_from_adjacency_matrix(adj_matrix)
  CC <- igraph::components(G_r)$membership

  ## Record distinct level sets
  if (i == 1 || any(CC != clustertree[[i-1]]$cluster)){
    clustertree[[i]] <- list(cluster=CC, radius=r * alpha)
    i <- i + 1
  }
  if (length(clustertree) == n - 1) break
}
```

The section of code in R Example 0.1 above is very inefficient, requiring more than several seconds on remarkably small data sets, using commodity-grade hardware[6]. It's been stated that the algorithm itself is computationally infeasible as it "requires a combinatorial search over all possible paths connecting any two points in the mesh" [20]. To avoid iterating to determine the critical r, others have come up with different approximations of estimating the empirical cluster tree. One such approximation of these radii distances that the **Python** package DEnsity BAsed CLustering (DeBaCl) [20] algorithm uses is to calculate the k-NN density estimate using the volume of the x_i-centered spheres directly:

$$\hat{f}(x_i) = \frac{k}{n \cdot v_d \cdot r_k^d(x_i)}$$

This density estimate $\hat{f}(x_i)$ is then used along with a graph-based approach akin to the approach from R Example 0.1 to estimate the hierarchy. This is an alternative formulation of λ (see section 3 in [7]), and is also the strategy employed by the newer **Topological Data Analysis (TDA)** R package with its 'clusterTree' function. Consider the usage of this function from the **TDA** package in R Example 0.2, using the example data from the dbscan examples page in the **Flexible Procedures for Clustering (FPC)** package documentation [16].

R Example 0.2.

```
suppressMessages({ require("TDA"); require("microbenchmark") })
## Generate data: 3 clusters
set.seed(665544)
N <- 600
x <- cbind(runif(10, 0, 10)+rnorm(N, sd=0.2),
           runif(10, 0, 10)+rnorm(N, sd=0.2))
K <- 50

## Density clustering using knn
microbenchmark(TDA::clusterTree(x, K, density = "knn", Nlambda = N), times = 15L)

## Unit: seconds
##                                                expr      min       lq
##  TDA::clusterTree(x, K, density = "knn", Nlambda = N) 2.403383 2.422091
##      mean   median       uq      max neval
##  2.487246 2.469215 2.514771 2.778309    15
```

[6]Benchmarks performed on a 2014 MacBook Pro equipped w/ a 2.5 GHz Intel Core i7

Although the approximation may have theoretical merit in its own right, runtime performance suffers for larger data sets. There are also a number of other issues with this implementation; by default, the 'Nlambda' parameter is kept small (100) as a means of reducing the number of λ values to cut the density estimates at. This algorithm only approximates the RSL solution, the results are not returned in representation well known to the R community (such as in the form of a `dendrogram` or `hclust` object), and there are no readily available means of comparing the results to the output of alternative hierarchical clustering trees for the purpose of analysis without significant post-processing.

Improved cluster tree estimation motivated by applied density-based clustering

Two motivating algorithms that are not dependent on the computationally expensive kernels and yet directly relate to the cluster tree include the widely known algorithm DBSCAN [11] and its corresponding extension, OPTICS [1]. Both algorithms have been mentioned, albeit briefly, in relation to the cluster tree [33] [32] and are of particular interest to this problem as they conform to Hartigan's notion of a high-density cluster at a fixed level λ of the cluster tree[7]. Can the algorithmic scalability of density-based clustering algorithms, such as DBSCAN, and the associated spatial indexing and efficiency improvements that have been proposed with it, be applied to the problem of efficiently estimating the cluster tree?

Campello *et al* in 2013 [4] and again in 2015 [5] (see Prop. 1 in the 2013 paper and Prop. 3.4 in the 2015 paper) established a hierarchical version of DBSCAN called "HDBSCAN", and in doing so made a few interesting observations. They extended a simple idea originally introduced by the OPTICS algorithm by defining a symmetric version of kNN distance called "mutual reachability distance", defined as follows:

$$d_{mreach}(x_p, x_q) = \max\{d_{core}(x_p), d_{core}(x_q), d(x_p, x_q)\}$$

where $d_{core}(x_p)$ is defined as the "core distance" or the distance from x_p to its 'minPts' nearest-neighbor. While KC and SD do not explicitly mentioned DBSCAN in their recent work [6] [7], it's straight forward to notice that by setting $k = minPts$, the core distance of a point x corresponds to the closed ball $B(x, r)$ with radius r_k that contains k data points (including the point itself, as the same with DBSCAN).

This observation simplifies the problem of estimating the cluster tree, and the tools and procedures implemented by readily-available density-based clustering algorithms now become relevant. Namely, Campello *et al* noted that by applying single-linkage or *Kruskal's algorithm* to the distances computed in *mutual reachability space*, the resulting minimum spanning tree (MST) corresponds to the hierarchy of DBSCAN* clusters at every parameter setting of ϵ. A similar observation was noted by [7], "A further simplification is that the graphs G_r don't need to be explicitly created. Instead, the clusters can be generated directly using Kruskal's algorithm, as is done for single linkage."

To illustrate, consider the following simpler version of Algorithm 1:

R Example 0.3.

```
## 'Mutual Reachability Distance'
mrd <- mapply(function(i, j) max(c(r_k[i], r_k[j], l2_dist[i, j] * alpha)),
              row(l2_dist), col(l2_dist))
mrd <- as.dist(matrix(mrd, nrow=n, ncol=n))

## Campellos et al's observation: running SL on MRD produces DBSCAN* results
clustertree2 <- hclust(mrd, method = "single")

## Now cutting the hclust object produces equivalent cuts to iterative approach
identical <- rep(FALSE, length(clustertree))
for (i in seq_along(clustertree)){
  identical[i] <- all.equal(clustertree[[i]]$cluster,
                            cutree(clustertree2, h = clustertree[[i]]$radius))
}
all(identical)
```

```
## [1] TRUE
```

[7] For $\alpha = 1$, DBSCAN* [4] (DBSCAN excluding border points) conforms exactly to Hartigans notion of a high-density cluster. Note that this also implies DBSCAN* with $minPts = k$ and fixed ϵ corresponds to the CCs of G_r a fixed level λ.

This simplification of the problem is one such motivation that may allow the empirical cluster tree computation to be expressed in a scalable fashion. The vast literature available for efficiently computing the MST becomes applicable. One such optimization that a Python implementation of HDBSCAN uses, which is part of the popular **SciKit-learn** library [27], is a algorithm known as the 'Dual-Tree Borůvka' (DTB) approach, which reduces the complexity associated with computing all $\binom{n}{2}$ pairwise distances to a conjectured asymptotically fastest $O(N \log N)$ runtime [24]. The approach is a clever combination using Borůvka steps (from Borůvkas MST algorithm) together with the branch-and-bound properties of various space indexing trees to—when the upper bounds allow—prune unneeded neighbor queries, dramatically reducing the number of distance calculations needed to be computed.

RSL algorithm design

The original RSL algorithm from [6], along with Algorithm 2 from [7], and various alternative approaches such as the "Spatially Adaptive" RSL algorithm from [3] all follow a similar algorithmic framework[8]. Each RSL variant can generally be broken down into the following steps:

Given data $X_n = \{x_1, x_2, \ldots, x_n\}$ drawn i.i.d from some unknown density f on $\mathbb{R}^d$,

1. Compute the radius r_* of the closed ball $B(x, r)$ around each x_i, denoted $r_*(x_i)$ where $r_*(x_i) = \min\{r : B(x_i, r)$ that satisfies $r_*\}$
2. Construct a graph G_r with nodes $\{x_i : r_*(x_i) \leq r\}$
3. Compute the connection radius $\|x_i - x_j\| \leq R$:
   (a) If R can be expressed as metric, use DTB to recover critical radii $r \in R$
   (b) Otherwise, compute all pairwise R and iterate as in R Example 0.1
4. Track the connected components $\hat{\mathbb{C}}(r)$ (using a disjoint-set data structure)

$r_*(x_i)$ is used to generically refer to any function that computes the radius of a closed ball centered at x_i. Note that this does not always correspond to the radius r_k, as in [3]. The important note is that such algorithmic components of RSL algorithms and similar RSL variants can be broken up into a set of modular sub-tasks using existing tools and data structures that, collectively, allow for the creation of a systematic set of tools for estimating and extending the *empirical* cluster tree.

Specifically, consider the the case where the special case where the connection radius αr ([3] use R) can be expressed as a metric. The use of efficient spatial-indexing structures can be used for the subsequent radius queries. The aforementioned Dual-Tree Borůvka MST algorithm has been implemented in the **MLPACK** library [9], has been ported to R with **RcppMLPACK** [21], and supports the use of arbitrary metrics through the use of template meta-programming. A more involved, but nonetheless viable alternative would be to augment the more mature ANN library [25], which is used for kNN queries in both the **FRNN** and **dbscan** packages, to support non-Minkowski metrics[9] The tracking of each connected component $\hat{\mathbb{C}}(r)$ can be efficiently performed using a disjoint-set data structure [34]. It is straightforward to compute the MST directly over the dense graph of all possible pairwise connection distances R, but such an algorithm *requires* the results be constrained *during the algorithms run time* to include edges iff $\|x_i - x_j\| \geq \max(r_*(x_i), r_*(x_j))$. Note the only reason the default MST computation was valid for Campello *et al*'s Hierarchical DBSCAN* [4] was due to the lack of setting of α.

Deliverables

The simplicity of the MST method on the transformed space of mutual reachability distances only follows from the RSL results if $\alpha = 1$, the consistency of which is an open problem. **Implementing the RSL algorithm that supports varying the setting of α would represent the first R implementation of an efficient cluster tree estimator following KC and SD's work**. Thus, the deliverables would be as follows:

1. Validated solution of Robust Single Linkage algorithm in **Rcpp**.

[8]The spatially adaptive RSL algorithm denotes αr as the connection radius R and replaces the radius of a point $r_k(x_i)$ with an alternative 'V-ball' representing the volume of a ball centered at x_i on a known manifold.

[9]Note that the ANN library actually uses *incremental distance calculations* [2] for the many of the internal distance calculations and requires recompilation of the library to support alternative measures, which is why it is mentioned as a 'more involved' alternative.

2. Capability of representing cluster tree using standard hierarchical representations (`dendrogram`, `hclust` objects).
3. Support for approximating kNN and mutual-kNN on arbitrary metric spaces (see [7]).
4. Demonstratable use of extension(s) to the cluster tree with the package (such as runt pruning, merge distortion, etc.).
5. Additional visualization and utility tools that help analyze the cluster tree. This includes contour plots, S3 specializations for `dendrogram` objects, a split tree options (see Section 7 discussing related work in [10]).
6. A unified R package for creating, analyzing, and visualizing empirical approximations of the cluster tree.

Qualifications

Motivated by the ingenuity of the hierarchical clustering extraction algorithm proposed by OPTICS—which could actually be considered a (very loose) approximation to RSL—I contributed to and became a coauthor of the popular **dbscan** package [13], a package developed to offer *significantly* faster versions of density-based algorithms in the DBSCAN family compared to native R implementations (such as the those in the **FPC**). In doing so, I gained a familiarity with both the algorithmic and theoretical components related to the popular density-based clustering algorithms DBSCAN, OPTICS, and HDBSCAN, along with general density and hierarchical-type clustering algorithms. As the author of the HDBSCAN implementation, which effectively represents the RSL solution with $\alpha = 1$ and $k = minPts$, I have demonstrably implemented a variant of RSL whose performance are scalability is essentially bounded by the 'dist' method from the **stats** package. More specifically, I have actually implemented a number of the basic data structures and algorithms required to make this project reality. Consider the following benchmarks using the same data set and parameter (K) from R Example 0.2:

```
suppressMessages({library("dbscan", quietly = T)})

## DBSCAN's performance depends the distance threshold, eps
dist_x <- dist(x)
eps_vals <- quantile(dist_x)[2:5]
microbenchmark(dbscan::dbscan(x, eps = eps_vals[1], minPts = K))

## Unit: milliseconds
##                                              expr      min      lq
##  dbscan::dbscan(x, eps = eps_vals[1], minPts = K) 1.573344 1.62555
##      mean   median       uq      max neval
##  1.755586 1.745274 1.825972 2.646154   100

microbenchmark(dbscan::dbscan(x, eps = eps_vals[2], minPts = K))

## Unit: milliseconds
##                                              expr      min       lq
##  dbscan::dbscan(x, eps = eps_vals[2], minPts = K) 2.633905 2.731498
##      mean   median       uq      max neval
##  2.906061 2.833576 2.952692 4.862466   100

microbenchmark(dbscan::dbscan(x, eps = eps_vals[3], minPts = K))

## Unit: milliseconds
##                                              expr      min       lq
##  dbscan::dbscan(x, eps = eps_vals[3], minPts = K) 3.728351 3.948377
##      mean   median       uq      max neval
##  4.420617 4.186122 4.612718 6.424246   100

microbenchmark(dbscan::dbscan(x, eps = eps_vals[4], minPts = K))

## Unit: milliseconds
##                                              expr      min      lq
##  dbscan::dbscan(x, eps = eps_vals[4], minPts = K) 4.565938 4.71685
##      mean   median       uq      max neval
##  4.995963 4.805423 5.038913 7.596683   100
```

```
## HDBSCAN is DBSCAN* at every n-1 eps levels,
## (in this case, equivalent to running DBSCAN* 599 times)
microbenchmark(dbscan::hdbscan(x, minPts = K, xdist = dist_x))
```

```
## Unit: milliseconds
##                                          expr      min       lq     mean
##  dbscan::hdbscan(x, minPts = K, xdist = dist_x) 21.93835 25.58014 30.89045
##    median       uq      max neval
##  27.36258 28.87601 108.3182   100
```

This was made possible due to a number of optimizations made to the internal components associated with the HDBSCAN computation, developed after discovering that many native R methods were inefficient. This includes, for example, computation of the MST, the conversion utilities related to `dendrogram` and `hclust` objects, etc.

```
suppressMessages({ library("optrees", quietly = T);
                   library("ape", quietly = T);
                   library("vegan", quietly = T) })

dist_m <- as.matrix(dist_x)
edge_list <- cbind(col(dist_m)[lower.tri(dist_m)],
                   row(dist_m)[lower.tri(dist_m)],
                   as.vector(dist_x))
 ## optree package
microbenchmark(optrees::msTreeKruskal(seq(attr(dist_x, "Size")), edge_list), times = 15L)
```

```
## Unit: seconds
##                                                    expr     min
##  optrees::msTreeKruskal(seq(attr(dist_x, "Size")), edge_list) 1.17099
##       lq     mean   median       uq      max neval
##  1.258867 1.286953 1.271425 1.319571 1.369981    15
```

```
microbenchmark(optrees::msTreePrim(seq(attr(dist_x, "Size")), edge_list), times = 15L)
```

```
## Unit: seconds
##                                                  expr      min     lq
##  optrees::msTreePrim(seq(attr(dist_x, "Size")), edge_list) 18.07935 18.573
##     mean   median       uq      max neval
##  19.12181 19.20862 19.56281 19.91762    15
```

```
microbenchmark(optrees::msTreeBoruvka(seq(attr(dist_x, "Size")), edge_list), times = 15L)
```

```
## Unit: seconds
##                                                    expr      min
##  optrees::msTreeBoruvka(seq(attr(dist_x, "Size")), edge_list) 66.04695
##       lq     mean   median       uq      max neval
##  66.67851 67.48934 67.19317 68.28682 69.92305    15
```

```
## iGraph
graph <- igraph::graph_from_adjacency_matrix(as.matrix(dist_x), weighted = T)
microbenchmark(igraph::mst(graph), times = 15L)
```

```
## Unit: milliseconds
##                expr      min       lq     mean   median       uq     max
##  igraph::mst(graph) 251.7339 293.3739 356.1521 379.6413 389.9387 513.048
##  neval
##     15
```

```
## ape package
microbenchmark(ape::mst(dist_x), times = 15L)
```

```
## Unit: seconds
##              expr      min       lq     mean   median      uq      max
##  ape::mst(dist_x) 27.62657 28.07033 28.17803 28.13589 28.3813 28.59785
##  neval
##     15
```

```
## vegan package
microbenchmark(vegan::spantree(dist_x), times = 15L)
```

```
## Unit: milliseconds
##                     expr      min       lq     mean  median      uq
##  vegan::spantree(dist_x) 1.618073 1.807647 2.065632 2.03914 2.30635
##       max neval
##  2.587063    15
```

```
## mst used by dbscan package (written by the author)
microbenchmark(dbscan:::prims(dist_x, attr(dist_x, "Size")), times = 15L)
```

```
## Unit: microseconds
##                                         expr     min       lq     mean
##  dbscan:::prims(dist_x, attr(dist_x, "Size")) 976.644 1094.588 1284.379
##    median       uq      max neval
##  1274.286 1392.777 1844.541    15
```

```
## Example of a simple conversion utility against native R S3 method
sl_x <- hclust(dist_x, method = "single")
microbenchmark(stats::as.dendrogram(sl_x))
```

```
## Unit: milliseconds
##                        expr     min       lq     mean   median       uq
##  stats::as.dendrogram(sl_x) 11.9797 12.52527 13.97469 13.05298 14.13366
##       max neval
##  25.65983   100
```

```
microbenchmark(dbscan:::as.dendrogram.hclust(sl_x))
```

```
## Unit: microseconds
##                                  expr     min     lq     mean   median
##  dbscan:::as.dendrogram.hclust(sl_x) 556.195 627.68 713.6239 714.9375
##        uq     max neval
##  780.6905 943.102   100
```

My experience contributing to the dbscan has also allowed me to become proficient with developing, optimizing, and documenting package-ready code for R.

As a coauthor with experience creating and comparing various kNN data structures in applied contexts [29] [28], I have experience developing, augmenting, and extending such spatial-indexing structures. I believe this experience would prove useful in working with the various indexing-trees offered by the **ANN** and **MLPACK** libraries.

Finally, as a data scientist[10] and motivated student of statistics, I am comfortable with the theory and terminology present in most of the cluster tree literature to implement all of the above deliverables under the guidance of my mentors, Michael Hahsler and Mikhail Belkin.

Timeline

This week-by-week timeline provides a rough guideline of how the project will be done.

May 15 – May 29:
Prepare initial report to mentors detailing the coding plan, and ensure the practical plan of approach matches mentors expectations.

May 30 – June 15:
Prepare initial skeleton of library, along with test data sets and needed libraries. Add initial standard data structures to use in the algorithm (disjoint set class, MST to `hclust` methods, `dendrogram` conversion methods, etc.). Rcpp connections with RcppMLPack added to access standard data structures and algorithms not immediately cluster tree specific, including the kNN tree structures and EMST implementation.

June 16 – June 26:
Establish initial validation tests using igraph. Finish initial implementation of basic RSL algorithm using Rcpp with variable α. Prepare evaluations.

[10] For my full curriculum vitae, see my web page: `http://mattpiekenbrock.com`

June 26 − July 15:
Begin work on testing and incorporating other metric spaces. Begin development the extensible API for augmenting the cluster tree approximation using an existing extension as a proof of concept (for example, merge distortion, or spatially adaptive radii calculations). Create initial round of unit tests and populate documentation.
July 15 − July 30:
Validate work with incorporating other metric spaces (testing the incorporation of the **MLPACK** extensions into the work flow). Include the option to compute the MST for arbitrary metrics defined using templates, if compatible. Add tests and documentation/examples of extending the tree. Develop the ability to compare different cluster trees using arbitrary tree comparison functions.
August 1 − August 15:
Ensure both kNN and mutual kNN graph can be computed using Algorithm 2 in [7] using the workflow above. Further refine tests and documentation for the whole project. Add visualization capability beyond the default options available from the `hclust` and `dendrogram` S3 plot methods from the **stats** package.
August 15 − August 30:
Check final documentation and ensure there are no programmatic issues. Write examples highlighting the use cases and differences between RSL-based approaches with traditional hierarchical clustering using other types of linkage criterion.

Management of Coding Project

The validation code for ensuring the optimized RSL implementation is given in R Example 0.1. I plan to commit every day. Minimally, I should have a commit at least every once every three days; as the theory behind the cluster tree is fairly involved, commits concerning existing cluster tree extensions may take more time to implement. Furthermore, the code within the **MLPACK** library is newer C++ code, augmenting the existing Dual-Tree Borůvka algorithm to work with the RSL scheme may also take longer to implement.

Schedule Conflicts

As of March 30, 2017, I have no other commitments. I have applied and awaiting responses from IBM and DataONE, however in the case of conflicts, GSOC takes priority over any other potential offers.

References

[1] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. Optics: ordering points to identify the clustering structure. In *ACM Sigmod record*, volume 28, pages 49–60. ACM, 1999.

[2] S. Arya and D. M. Mount. Algorithms for fast vector quantization. In *Data Compression Conference, 1993. DCC'93.*, pages 381–390. IEEE, 1993.

[3] S. Balakrishnan, S. Narayanan, A. Rinaldo, A. Singh, and L. Wasserman. Cluster trees on manifolds. In *Advances in Neural Information Processing Systems*, pages 2679–2687, 2013.

[4] R. J. Campello, D. Moulavi, and J. Sander. Density-based clustering based on hierarchical density estimates. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 160–172. Springer, 2013.

[5] R. J. Campello, D. Moulavi, A. Zimek, and J. Sander. Hierarchical density estimates for data clustering, visualization, and outlier detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 10(1):5, 2015.

[6] K. Chaudhuri and S. Dasgupta. Rates of convergence for the cluster tree. In *Advances in Neural Information Processing Systems*, pages 343–351, 2010.

[7] K. Chaudhuri, S. Dasgupta, S. Kpotufe, and U. von Luxburg. Consistent procedures for cluster tree estimation and pruning. *IEEE Transactions on Information Theory*, 60(12):7900–7912, 2014.

[8] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 24(5):603–619, 2002.

[9] R. R. Curtin, J. R. Cline, N. P. Slagle, W. B. March, P. Ram, N. A. Mehta, and A. G. Gray. MLPACK: A scalable C++ machine learning library. *Journal of Machine Learning Research*, 14:801–805, 2013.

[10] J. Eldridge, M. Belkin, and Y. Wang. Beyond hartigan consistency: Merge distortion metric for hierarchical clustering. In *COLT*, pages 588–606, 2015.

[11] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[12] C. Fraley and A. E. Raftery. Model-based clustering, discriminant analysis, and density estimation. *Journal of the American statistical Association*, 97(458):611–631, 2002.

[13] M. Hahsler and M. Piekenbrock. *dbscan: Density Based Clustering of Applications with Noise (DBSCAN) and Related Algorithms*, 2017. R package version 1.1-0.

[14] J. A. Hartigan. Consistency of single linkage for high-density clusters. *Journal of the American Statistical Association*, 76(374):388–394, 1981.

[15] J. A. Hartigan and J. Hartigan. *Clustering algorithms*, volume 209. Wiley New York, 1975.

[16] C. Hennig. *fpc: Flexible Procedures for Clustering*, 2015. R package version 2.1-10.

[17] A. Hinneburg, D. A. Keim, et al. An efficient approach to clustering in large multimedia databases with noise. In *KDD*, volume 98, pages 58–65, 1998.

[18] A. K. Jain and R. C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[19] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

[20] B. P. Kent, A. Rinaldo, and T. Verstynen. Debacl: A python package for interactive density-based clustering. *arXiv preprint arXiv:1307.8136*, 2013.

[21] Q. Kou. Rcppmlpack: R integration with mlpack using rcpp. 2016.

[22] S. Kpotufe and U. von Luxburg. Pruning nearest neighbor cluster trees. *arXiv preprint arXiv:1105.0540*, 2011.

[23] H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek. Density-based clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(3):231–240, 2011.

[24] W. B. March, P. Ram, and A. G. Gray. Fast euclidean minimum spanning tree: algorithm, analysis, and applications. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 603–612. ACM, 2010.

[25] D. M. Mount and S. Arya. Ann: library for approximate nearest neighbour searching. 1998.

[26] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[27] 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.

[28] M. Piekenbrock, J. Robinson, L. Burchett, S. Nykl, B. Woolley, and A. Terzuoli. Automated aerial refueling: Parallelized 3d iterative closest point: Subject area: Guidance and control. In *Aerospace and Electronics Conference (NAECON) and Ohio Innovation Summit (OIS), 2016 IEEE National*, pages 188–192. IEEE, 2016.

[29] J. Robinson, M. Piekenbrock, L. Burchett, S. Nykl, B. Woolley, and A. Terzuoli. Parallelized iterative closest point for autonomous aerial refueling. In *International Symposium on Visual Computing*, pages 593–602. Springer, 2016.

[30] J. Sander. Density-based clustering. In *Encyclopedia of Machine Learning*, pages 270–273. Springer, 2011.

[31] SIGKDD. Sigkdd news : 2014 sigkdd test of time award. `http://www.kdd.org/News/view/2014-sigkdd-test-of-time-award`, 2014. (Accessed on 10/10/2016).

[32] A. Singh, C. Scott, R. Nowak, et al. Adaptive hausdorff estimation of density level sets. *The Annals of Statistics*, 37(5B):2760–2782, 2009.

[33] W. Stuetzle and R. Nugent. A generalized single linkage method for estimating the cluster tree of a density. *Journal of Computational and Graphical Statistics*, 19(2):397–418, 2010.

[34] R. E. Tarjan. *Data structures and network algorithms*. SIAM, 1983.

[35] D. Wishart. Mode analysis: A generalization of nearest neighbor which reduces chaining effects. *Numerical taxonomy*, 76(282-311):17, 1969.

[36] M. A. Wong and T. Lane. A kth nearest neighbour clustering procedure. In *Computer Science and Statistics: Proceedings of the 13th Symposium on the Interface*, pages 308–311. Springer, 1981.