THE UNIVERSITY *of* EDINBURGH

Edinburgh Research Explorer

HEADSS: HiErArchical Data Splitting and Stitching software for non-distributed clustering algorithms

Citation for published version:
Crake, DA, Hambly, NC & Mann, RG 2023, 'HEADSS: HiErArchical Data Splitting and Stitching software for non-distributed clustering algorithms', *Astronomy and Computing* , vol. 43, 100709, pp. 1-9.
https://doi.org/10.1016/j.ascom.2023.100709

Digital Object Identifier (DOI):
10.1016/j.ascom.2023.100709

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Publisher's PDF, also known as Version of record

Published In:
Astronomy and Computing

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 02. May. 2023

Astronomy and Computing 43 (2023) 100709

Contents lists available at ScienceDirect

Astronomy and Computing

journal homepage: www.elsevier.com/locate/ascom

Full length article

HEADSS: HiErArchical Data Splitting and Stitching software for non-distributed clustering algorithms

D.A. Crake *, N.C. Hambly, R.G. Mann

Institute for Astronomy, University of Edinburgh, Royal Observatory Edinburgh, Blackford Hill, Edinburgh, EH9 3HJ, United Kingdom

ARTICLE INFO

Article history:
Received 3 November 2022
Accepted 15 March 2023
Available online 8 April 2023

MSC:
0000
1111

Keywords:
Methods: Data analysis
Methods: Statistical
Methods: Miscellaneous
Methods: Numerical

ABSTRACT

The increase in data volume is challenging the suitability of non-distributed and non-scalable algorithms, despite advancements in hardware. An example of this challenge is clustering. Considering that optimal clustering algorithms scale poorly with increased data volume or are intrinsically non-distributed, accurate clustering of large datasets is increasingly resource-heavy, relying on substantial and expensive compute nodes. This scenario forces users to choose between accuracy and scalability. In this work, we introduce HiErArchical Data Splitting and Stitching (HEADSS), a Python package designed to facilitate clustering at scale. By automating the splitting and stitching, it allows repeatable handling, and removal, of edge effects. We implement HEADSS in conjunction with HDBSCAN, where we achieve orders of magnitude reduction in single node memory requirements for both non-distributed and distributed implementations, with the latter offering similar order of magnitude reductions in total run times while recovering analogous accuracy. Furthermore, our method establishes a hierarchy of features by using a subset of clustering features to split the data.[1]

© 2023 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

As a result of continuous increases in data volume, scalable and distributed analysis techniques are essential. The plethora of incoming data affects all aspects of our lives, from personal data to scientific research. An illustration of this is in the field of astronomy. Over the last two decades, the volume of incoming data has scaled from a few Terabytes to hundreds of Petabytes, increasing to Exabytes over the next few years (Zhang and Zhao, 2015). In many fields, it is insightful to cluster these datasets for a range of applications including but not limited to outlier detection, pattern recognition and even physical clustering.

Unfortunately, clustering algorithms require significant computational resources as, generally, each data point must be evaluated against every other data point resulting in a complexity factor of $O(n^2)$, which is unsuitable for big data analysis. Clustering big data poses many challenges and no single approach performs well against all evaluation metrics (Ajin and Kumar, 2016). K-means clustering scales favourably with complexity $O(nk)$, where n is the number of data points and k is the number of clusters (Na et al., 2010). There are also works that improve the scalability of K-means clustering through the construction of coresets in Lucic et al. (2016), or a distributed implementation in Balcan et al. (2013). However, K-means clustering has certain characteristics which can limit its suitability for certain datasets. The most prominent constraint is that the number of centroids must be defined, requiring a known number of clusters before analysis. Furthermore, K-means clustering is a form of the Expectation Maximisation (EM) algorithm (Moon, 1996), which is optimal for spherical clusters. There are approaches to reduce the impact of these limitations, but none fully eradicate these fundamental issues (Singh et al., 2011).

Another algorithm to consider is DBSCAN, which can identify an unspecified number of arbitrary-shaped clusters of specific density. The full algorithm is outlined in Ester et al. (1996) and solves the major hurdles in K-means clustering. DBSCAN defines clusters by analysing the neighbouring points within a defined distance (ϵ), and if the number of neighbours exceeds the selected minimum cluster size the object becomes a central cluster point. This process is repeated until the edge of the cluster is defined, otherwise, the object is classed as noise. However, the rigidity of ϵ reduces overall accuracy as DBSCAN cannot identify clusters of varying densities. Furthermore, DBSCAN has a complexity issue, even in the best case, it cannot achieve sub-$O(n \log n)$, with the worst-case scenario reaching $O(n^2)$. An issue exacerbated by the fact the full dataset must be loaded to memory and evaluated by a single node (Ali et al., 2010). McInnes et al. (2017) presents a Hierarchical Python implementation, known as HDBSCAN, allowing the identification of varying cluster densities through various

* Corresponding author.
E-mail address: dennis.crake@ed.ac.uk (D.A. Crake).

[1] Source code and examples are available at https://github.com/D-Crake/HEADSS

https://doi.org/10.1016/j.ascom.2023.100709
2213-1337/© 2023 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

values of ϵ. Nevertheless, HDBSCAN continues to have the same heavy computational requirements as the standard DBSCAN. The complexity limitations of big data clustering with DBSCAN and the limitations of K-means clustering discussed above force users to decide between accuracy and scalability.

As clustering is a common task, there are methods for scaling specific algorithms within the literature. Such examples include methods for scaling Hierarchical Agglomerative Clustering (Sumengen et al., 2021), linkage-based hierarchical algorithms (Bateni et al., 2017) and several k-means or centroid-linkage algorithms in Lattanzi et al. (2020).

The astrophysical challenge that inspired the current work is identifying stellar clusters within the Milky Way. Clustering 5−dimensional stellar data (three positional and two velocity measurements), represents the first full-scale result obtained by HEADSS. The full details are to be described by Crake et al. (In Prep). The challenge represents an ideal dataset due to relatively small clusters than the total feature space and is representative of numerous physical clustering challenges. Research revealed a lack of easily repeatable approaches within the wider scientific community, with HEADSS aiming to correct this through a user-friendly package.

1.1. Splitting of data

A simple method to avoid the complexity and memory requirements is partitioning the data into manageable regions known as *"partitions"* henceforth. Partitions are comparable to the concept of *canopies* for clustering and *"blocking"* in record linkage clustering, which identifies pairs of records that represent the same entity. Both techniques act as a preprocessing strategy designed to reduce the number of comparisons. *Blocking* acts as a preprocessing filter to partition data to avoid comparisons between distinctly unrelated records reducing computational requirements. Record linkage is commonly used for matching entries between catalogues and the blocking criteria can be trained through supervised techniques using a set of labelled datasets, as described in Michelson and Knoblock (2006) or identified by unsupervised methods as described in O'Hare et al. (2019). Canopy clustering can be considered a density-based analogue to HEADSS. The process of canopy creation is as follows:

- Select a random data point not associated with a canopy to act as a new canopy centre.
- All data points within a distance, $T1$, are part of the canopy.
- All data points within a greater distance, $T2$, are associated with the canopy but may join other canopies.
- Repeat these steps until all data is within a canopy.

The effectiveness of canopy clustering relies on suitable parameters and is subject to dimensionality issues (Kumar et al., 2014; Sagheer and Yousif, 2021; McCallum et al., 2000). The approach of partitioning data using either of these methods or HEADSS allows the use of complex algorithms on big data with the same hardware. Nonetheless, partitioning can cause issues to arise, most often on the extremities of a cluster, with several possible edge effects arising.

For this work, we define edge effects as any artefact, loss of group members or inconsistency introduced by partitioning the feature space that would otherwise not be present if the complete dataset was analysed. Additionally, an edge effect can be defined as any change in result that disproportionately affects a specific region of the feature space, i.e. along the threshold of a partition. Typically, edge effects are caused by clusters spanning multiple partitions leading to potential data loss. As HDBSCAN requires a minimum cluster size to be defined, the partial clusters are at risk of dropping below the threshold resulting in incompleteness resulting in clusters with non-physical boundaries. Conversely, clusters large enough to be partially identified in multiple partitions are also problematic, as accurately differentiating valid mergers from neighbouring clusters is challenging at scale.

The prevention and handling of edge effects that occur from splitting data are the fundamental issues this work aims to address by setting a standard approach. Our software, HiErArchical Data Splitting and Stitching (HEADSS henceforth), provides both the position for cuts and stitching boundaries that maximises the distance of any given point from an edge boundary, eradicating edge effects in a range of representative datasets and allowing compatibility with all clustering algorithms. Additionally, the software provides functionality to handle the split, clustering and stitching process using HDBSCAN as an example clustering algorithm. Furthermore, where individual clusters span a significant fraction of the feature space, it handles the processing mergers defined by three hyperparameters. HEADSS is an algorithm that works in conjunction with existing clustering algorithms and works with a variety of common clustering algorithms, ensuring the clustering of big data remains accurate, reliable and repeatable with no direct user judgement required during the splitting or stitching phase.

2. Introducing HEADSS

In this Section, we outline the basic principles and assumptions of HEADSS. This work reduces complexity by partitioning the data, reducing the number of objects clustered in a given partition. We refer to this process as "splitting". When splitting the data by a subset of features, this naturally establishes a hierarchy. This work splits data into evenly sized partitions across four complementary layers. Due to the symmetry of the feature space and the repeating patterns, up to four layers are required for any given N, where N defines the number of partitions in the base layer for each feature to be split. The base layer represents a rudimentary partition by a simple N^D grid, where N is the number of splits in each feature and D is the number of features used for splitting. Hence, for 2 features, the feature space is split into a $N \times N$ grid. This establishes multidimensional partitions with each length equal to FR/N, where FR refers to the "Full Range" of a given feature in the whole dataset. The Secondary layer offsets the base layer by $FR/2N$, creating a $(N-1)^D$ grid, centred where four partitions intersect in the base layer. The Tertiary layer partitions centre where two base layer partitions meet perpendicular to the nearest axis. Similarly, the Quaternary layer partitions centre where two base layer partitions meet parallel to the nearest axis. A 2D visualisation of the layers for $N = 2, 3$ & 4 respectively can be seen in Fig. 1. Since the clustering partitions occupy an equal fraction of the total feature space, we assume the data does not predominantly occupy a small area of the full feature space.

The amalgamation of these layers allows a single feature space where any point resides at least FR/2N distant from any boundary. HEADSS also establishes the splitting layer that maximises the distance from any boundary for the full feature space, a process we refer to as "stitching". This process selects cluster centroids that maximise distance from a boundary while dropping repeated clusters. For the selected centroids, the full cluster remains including the individual members that span beyond the stitching boundary. An assumption at this stage is that clusters do not span a large fraction of the total feature space (specifically less than $FR/2N$). By ensuring the centroid, not the members, resides as far from a cut as possible, we minimise the occurrence of partial clusters and avoid a single point occupying multiple clusters provided our assumptions hold for the underlying distribution. Fig. 2 visualises the optimal stitching partitions for the $N = 2, 3$ & 4

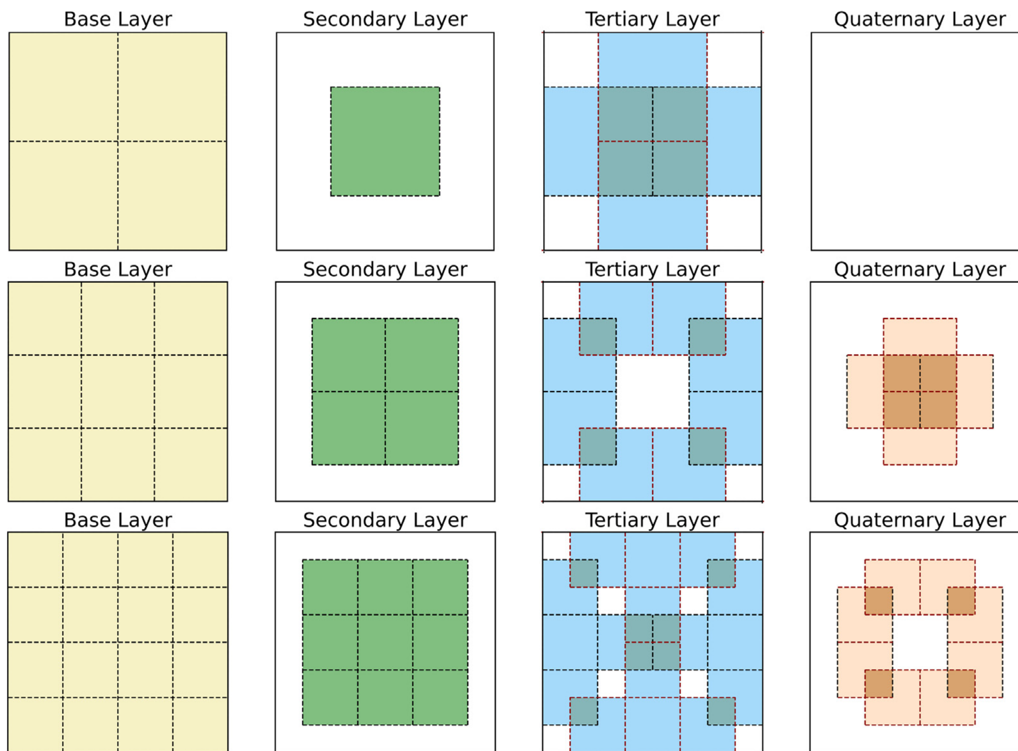
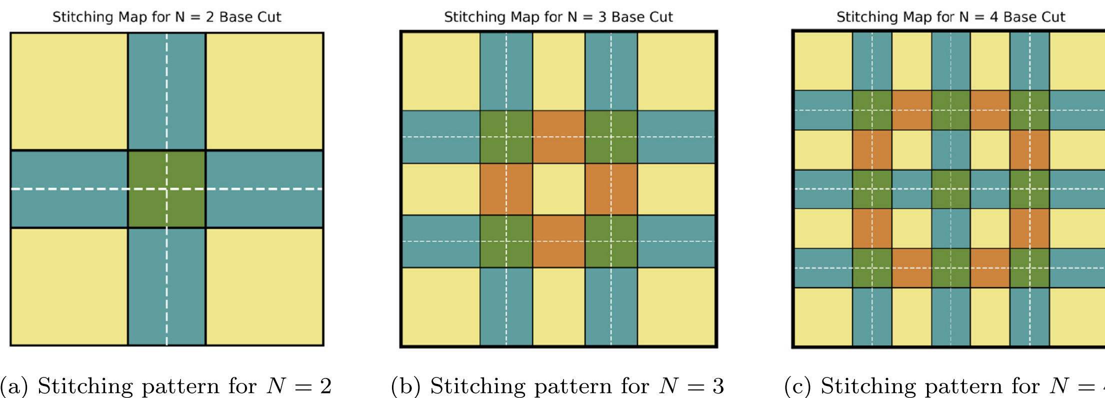


Fig. 1. 2D Visualisation of the splitting layers for $N = 2, 3$ & 4 base layer (Top to bottom). For $N = 2$ the Quaternary layer remains unused. Darker colours indicate overlapping partitions in a single layer.



(a) Stitching pattern for $N = 2$ (b) Stitching pattern for $N = 3$ (c) Stitching pattern for $N = 4$

Fig. 2. Visualisations of 2D stitching partitions with $N = 2, 3$ & 4 (left to right). The colour of each partition represents the splitting layer which maximises the distance to any boundary. The white dashed line indicates the base layer cuts for reference.

splits seen in Fig. 1. The central partitions cover identical fractions of feature space, while the outer partitions extend to the edge of the full feature space due to a lack of boundaries introduced by the splitting process.

Minimising the number of cuts and dimensionality (features) minimises overhead requirements and artefacts. HEADSS has a complexity of $O(N_{partitions})$, where $N_{partitions}$ is the number of partitions. $N_{partitions}$ scales as $(2N - 1)^D$, where N = number of base layer cuts and D = number of features. The complexity scales unfavourably with the number of features (D), but remains reasonable for N. The extent of scaling is visualised in Fig. 3, compounding the importance of reducing dimensionality. As each partition is analysed independently, the clustering can occur in parallel across multiple nodes, whereas the splitting and stitching can also be distributed with software such as Apache Spark.

In summary, our algorithm works as follows:

- Split the data into evenly sized partitions across four complementary layers, see Fig. 1.
- Cluster partitions independently using user selected clustering approach.
- Identify duplicate clusters and select the partition that maximises cluster centroid from an edge, see Fig. 2.

under the assumption that the clusters do not span a large fraction of the feature space ($< FR/2N$) and a small number of clustering features (or a subset of features) exists to partition the data.

In the following sections, we shall show examples of the HEADSS API before exploring the performance of our algorithm across a range of test datasets selected to challenge our assumptions, showcase the performance and highlight the results when the assumptions do not hold. In Section 5, we demonstrate a modified workflow that handles clusters that do span a significant fraction of the feature space.

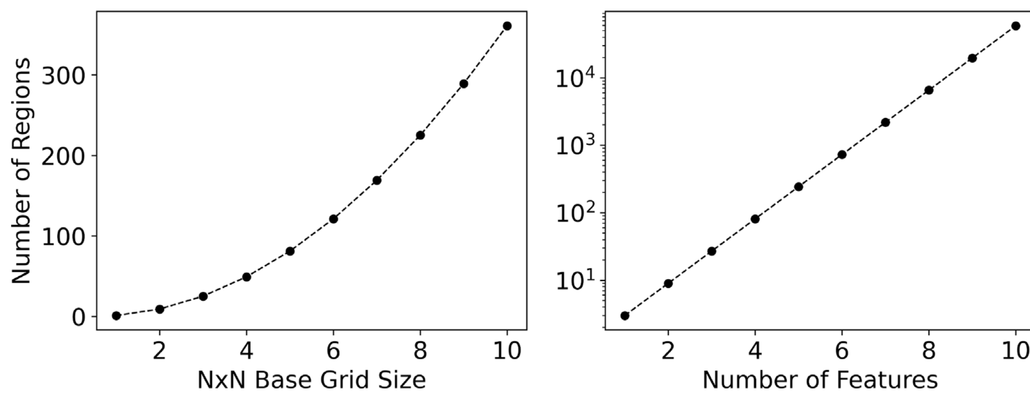


Fig. 3. Scaling of the number of partitions required for increasing base layer size with 2 features (left) and number of features with an $N = 2$ base layer (right) highlighting the importance of establishing a small subset of features for the splitting process.

2.1. Quick start guide

The GitHub, linked above, contains a series of user guides, including a quick start guide and all the code required to reproduce the analysis in this work. However, there are two major use cases to consider, those that use the integrated clustering algorithm and those which use alternative clustering algorithms which require greater user input but offer greater flexibility, such as running partitions concurrently. Currently, HDBSCAN is the sole integrated algorithm, with analysis performed with the following code:

```
1  import numpy as np
2  import pandas as pd
3  import HEADSS
4
5  # import full dataset for clustering
6  data = pd.read_csv(filepath)
7
8  # Perform split, clustering & merge using HEADSS.
9  merge = headss_merge(df = data, N = 2, merge = False
       ,
10                      split_columns = ['col1', 'col2'
       ],
11                      cluster_columns=['col1','col2'
       ],)
12
13 # clustering result returned as a pandas.DataFrame.
14 merged_df = merge.members_df
```

Listing 1: Simple HEADSS example.

All clustering parameters inherited from HDBSCAN and for merging, described in Section 5, are omitted but shown in the user guide. The API is designed for the integrated use case, whereas the functionality for splitting and stitching is shown in the user guides. The most common function will be retrieving the partitioning and stitching boundaries which can be obtained using a few lines of code:

```
1  import numpy as np
2  import pandas as pd
3  import HEADSS
4
5  # import full dataset for clustering
6  data = pd.read_csv(filepath)
7
8  # Calculate regions to split
9  head = headss_regions(N = 2, df = data,
10                      split_columns=['col1','col2'])
11
12 # Return DataFrame of data with partition id.
13 partitions = head.split_data
14
15 # Get partition boundaries
16 part_bound = head.split_regions
17
18 # Get stitch boundaries
19 stitch_bound = head.stitch_regions
```

Listing 2: HEADSS example for returning splitting and stitching boundaries.

This code showcases two splitting approaches; either allow HEADSS to assign a "partition" column to the entire database, or for particularly large datasets, *head.split_regions* provides the split boundaries. If the dataset is too large for memory, HEADSS can calculate the suitable partitions using the maximum and minimum values for each feature. This approach allows the user to split and cluster by partition, while the head.stitch_regions provides the boundaries for stitching regions for the resulting cluster centroids.

3. Example datasets

We have selected ten examples from Fränti and Sieranoja (2018) that demonstrate and push the limits of our software while representing various potential use cases. Fig. 4 shows the selected datasets. They range from the highly-dense *birch1* and *worms* with approximately 100,000 data points to the sparsely populated *flame*, *pathbased*, *spiral* and *jain* each with around 300 data points. This selection of datasets contains examples with numerous compact clusters while others contain sparse clusters that span a large fraction of the feature space. This work has been developed with the physical clustering of astronomical data in mind, where *a3, D31* and *birch1* are the most representative and satisfy our assumptions in Section 2. Our test datasets are introduced in one of the following papers Gionis et al. (2007), Kärkkäinen and Fränti (2002), Fu and Medico (2007), Chang and Yeung (2008), Veenman et al. (2002), Zhang and Zhao (2015), Jain and Law (2005), Karypis et al. (1999), Sieranoja et al. (2019).

4. Performance

To evaluate the performance of HEADSS, we need a baseline with no splitting of the data. Due to the reasons outlined earlier in Section 1, we select HDBSCAN throughout the remainder of this paper, but any algorithm can be implemented if preferred. Using the hyperparameters in Table 1, we identify the clusters shown in Fig. 5. We see good agreement with visual classifications for 7 of the datasets for the major clusters, with the exceptions being *worms*, *pathbased* & *jain*. *Worms* is a dataset with clusters of varying density and noise. Overall, the majority of clusters appear partially identified with a significant fraction lost as noise. *Jain* performs well separating the two curves, however, half of the

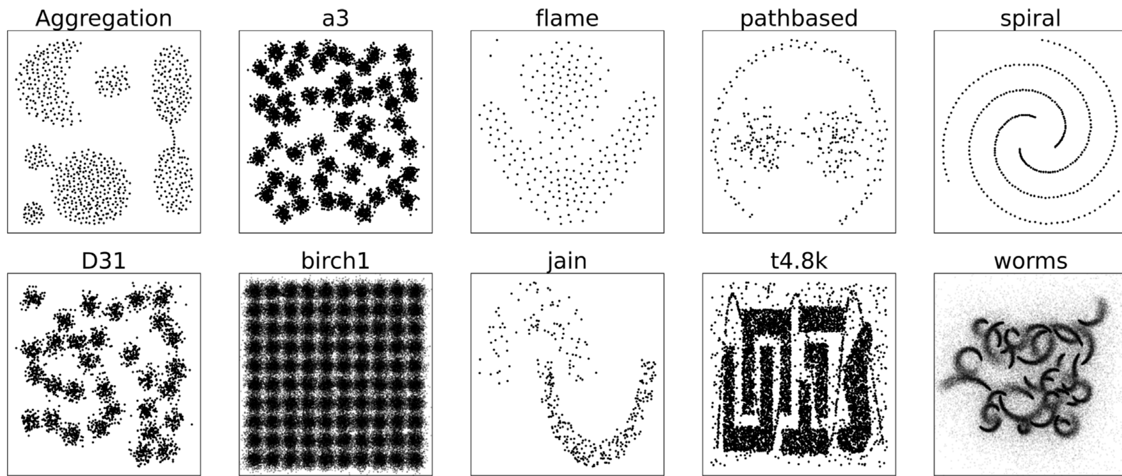


Fig. 4. Example datasets used to evaluate the performance of HEADSS in conjunction with HDBSCAN.

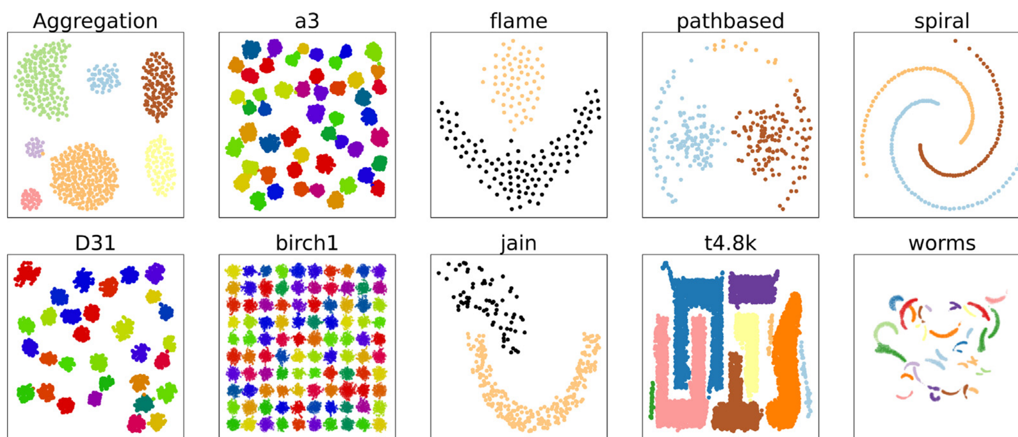


Fig. 5. Clusters identified by HDBSCAN on the full dataset.

Table 1
HDBSCAN hyperparameters for complete dataset clustering.

Dataset	min_samples	allow_single_cluster	cluster_method	min_cluster_size
Aggregation	20	False	leaf	20
a3	20	False	leaf	20
flame	5	False	leaf	20
pathbased	10	False	eom	5
spiral	10	False	leaf	5
D31	20	False	leaf	20
birch1	10	False	leaf	200
jain	1	False	eom	50
t4.8k	10	False	eom	20
worms	200	False	leaf	200

structure in the upper curve is lost due to low density. Finally, the worst performance by far is *pathbased*, the outer ring is poorly identified with sections merging with the central clusters.

Recall that we do not aim to improve on the baseline results in Fig. 5 as our aim is that HEADSS seamlessly assists implementation but does not impact the clustering results. We evaluate the performance for an $N = 2$ implementation due to the limited size of the example datasets. In this implementation, with two features, each partition covers 25% of the total feature space. In the datasets where clusters span a significant fraction $>$ 25% of the feature space, our assumptions do not hold, meaning we anticipate partial clusters occurring. To minimise data loss and account for the new projections, we adjust the clustering hyperparameters to those seen in Table 2.

In Fig. 6, the resulting clusters from the splitting and stitching process show promising results. Exploring the target datasets (*a3, D31* and *birch1*), we observe near-identical performance to the non-split clustering. Considering these best represent the astronomical data that inspired this work and optimal use cases, the basic splitting and stitching functions are suitable when the above assumptions hold. Nonetheless, to broaden the potential impact of this work the seven remaining datasets must also be considered.

The noisy *worms* dataset predominantly identifies the same clusters as the baseline as the clusters do not span a sufficient fraction of the feature space to cause edge effects. Initial impressions of the remaining six datasets are that the known clusters are broadly identified but with excessive edge effects. The

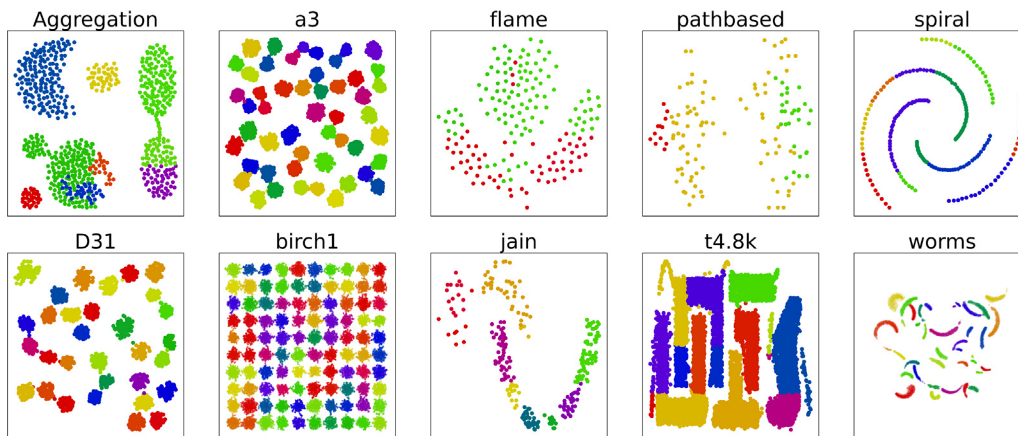


Fig. 6. Clusters identified by HDBSCAN with an N=2 implementation of HEADSS.

Table 2
HDBSCAN hyperparameters for N=2 HEADSS region clustering.

Dataset	min_samples	allow_single_cluster	cluster_method	min_cluster_size
Aggregation	10	False	eom	10
a3	20	False	leaf	20
flame	5	True	eom	5
pathbased	1	True	eom	30
spiral	1	True	leaf	3
D31	20	False	leaf	20
birch1	10	False	leaf	200
jain	1	False	eom	5
t4.8k	30	False	eom	30
worms	200	False	leaf	100

edge effects are directly caused by splitting the dataset and independently clustering each partition. Therefore, further evaluating these datasets is impossible without first correcting the significant edge effects.

5. Removing artefacts

As mentioned above, the remaining six datasets (*Aggregation, flame, pathbased, spiral, jain* and *t4.8k*) all show a significant deviation from the baseline clusters. Typically, identifying cluster mergers is extremely difficult and often requires considerable human input or sophisticated algorithms when partitioning the data. Human input signifies the process is not scalable nor repeatable, whereas a sophisticated algorithm can interfere with the clusters or require additional resources to be available. With HEADSS, potential mergers have a subset of mutual members from at least two independent splitting layers. This region of potential mutual members enables a numerical evaluation of similarity between neighbouring clusters.

The merging process applies as follows:

- Identify clusters that potentially overlap.
- Compare mutual members in overlapping clusters.
- Quantify defined overlapping parameters to determine suitable mergers.

This numerical evaluation requires a cross-match of members for all clusters that potentially overlap. As cross-matches are computationally expensive, it is vital to identify potential merges efficiently. HEADSS achieves this by evaluating the cluster limits for overlaps, which currently has complexity $O(k^2)$ where k is the number of clusters rather than the number of data points. From this point, we can cross-match only the partitions that potentially overlap defined by the split and stitch boundaries described in Section 2. This process creates three further hyperparameters which quantify merges; *"overlap_threshold"*, the fraction of mutual members within the overlap region, *total_threshold*, the fraction of mutual members within the whole cluster and *minimum_members*, the minimum mutual members to allow a merge. Each parameter protects from a few rogue points or small clusters falsely merging large sections of the feature space.

At this point, we have created a tree of cluster interactions with pair matches and chains of long clusters linking, despite not all interacting. An example of this is a horseshoe or crescent pattern such as those in *t4.8k*. By iterating the list of merger identifiers, rather than the members, we ensure all chains merge to a single identifier minimising the complexity of the merging process dramatically. Fig. 7 shows the final result from HEADSS with merging hyperparameters from Table 3.

HEADSS, with a subsequent merge, generally matches the baseline results in Fig. 5 for the majority of the datasets, particularly those with high density. Merging has performed best on *t4.8k*, successfully connecting the six dominant clusters. *Aggregation* also performs well, but the rightmost conjoined clusters have merged and a small amount of data loss in the largest cluster. The merge has improved the results for *spiral*, but merging cannot reverse the loss of data in two of the spiral arms caused by the splitting process. *Jain* is a unique case where the completeness is improved but the ability to identify the full cluster is reduced compared to the baseline. Splitting the data successfully identifies the lost region in the upper curve while splitting the lower curve into several smaller clusters. The lower curve again shows the splitting of different densities first highlighted in *a3*. The merge reduces the divisions across partition boundaries in the lower curve, but cannot merge clusters split by the differing densities. Finally, we have *flame* and *pathbased*. Neither return ideal performance, practically identifying a single large cluster due to their low density. The space between clusters is less defined when split into partitions, yielding clusters which span the gap causing the single cluster after merging. While those results are not ideal, neither dataset represents a realistic use case. For a sparse cluster

Table 3
Merging hyperparameters for test datasets.

Dataset	total_threshold	overlap_threshold	minimum_members
Aggregation	0.10	0.90	3.0
flame	0.10	0.50	1.0
pathbased	0.10	0.50	1.0
spiral	0.01	0.01	1.0
jain	0.10	0.70	1.0
t4.8k	0.10	0.50	10.0
worms	0.10	0.50	1.0

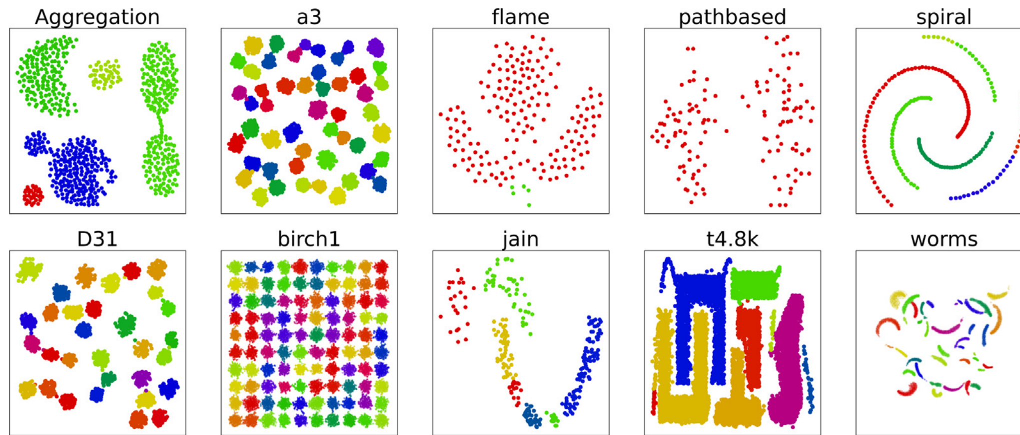


Fig. 7. Clusters identified by HDBSCAN with an $N = 2$ implementation of HEADSS including the merger capabilities.

to span a large fraction of the total feature space and remain identifiable, the dataset is either not very large (HEADSS would not be required) or obscured by other clusters and noise.

These results show the merging capability provides a stark improvement in the impact and suitability of HEADSS over the results in Fig. 6 where the merging capabilities are not used. Merging is best suited for clusters with well-defined boundaries, particularly dense clusters. Merging clusters in HEADSS is especially successful in two of the example datasets and somewhat in a further two, representing a wide range of potential use cases. The only use cases where merging was detrimental to the performance were designed to be challenging and do not represent a realistic clustering application where complexity is a concern. Considering these results are solely evaluated with HDBSCAN, we may improve performance on specific datasets with other clustering algorithms leading to further suitable use cases.

6. Computational performance

In addition to the standardisation of big data clustering, the ambition of the HEADSS software is to reduce the peak computational requirements. The two factors are the single-node memory requirements and computational runtime. In the following section, we shall explore the theoretical reduction of these factors for an optimal use case.

Peak memory requirements are dependent on the number of objects within a single clustering calculation. For HEADSS, the optimal dataset is evenly distributed across the entire feature space, meaning the number of objects with each partition is proportional to the fraction of feature space covered. In this scenario, HEADSS offers an expected reduction in memory requirements by a factor of $O(N^D)$, where N is again the number of splits in each feature, and D is the number of dimensions. Considering many clustering algorithms have complexity between $n\log(n)$ and n^2, where $n =$ number of objects. When $N = D = 2$ HEADSS offers a $\sim 4{-}16\times$ reduction in peak requirements, where n is the number of points to cluster. The exact factor of reduction depends on the clustering algorithm used, the number of splits and the overall distribution of the data. If the data densely populates a sub-region of the feature space, consider splitting across an alternative set of features. If no such features exist, further partitions in the dense regions or an alternative approach, such as canopy clustering, should be considered.

Theoretical runtimes depend on many factors, including but not limited to the choice clustering algorithm, the IO bandwidth and the dataset distributions. However, using empirical analysis, we can evaluate the impact of HEADSS on runtimes. The runtime of all clustering algorithms is dominated by the number of objects to cluster. As HEADSS facilitates the clustering of smaller partitions, for large datasets (number of objects $> 10^6$) the reduced clustering runtimes easily offset the additional overheads introduced. The top panel of Fig. 8 visualises the non-distributed runtimes of HEADSS compared to a baseline HDBSCAN implementation. The performance is for a 2-Dimensional clustering, with HEADSS becoming required from roughly 10^6 objects. Below this threshold, the overheads introduced by spitting and stitching the data account for a significant fraction of the runtime. Nonetheless, due to the short overall runtime, the increase for HEADSS runtimes is to the order of 10's seconds. Above the threshold, the increase in clustering runtimes negates the overhead, allowing comparable or even better runtimes as early as 10^7 objects, a comparatively small clustering task.

The analysis has shown that HEADSS successfully reduces peak memory requirements for a clustering task with comparable runtimes to HDBSCAN on a non-distributed deployment with reduced memory requirements. Yet, a substantial advantage of this package is the ability to run the independent clustering analysis concurrently, essentially distributing the clustering phase. In a deployment with concurrent clustering, the total runtime is determined by the most populated partition. The theoretical possible runtimes are shown in the bottom figure of Fig. 8, which is the top plot scaled by a factor of N^D, which introduces an assumption that the data is evenly distributed and all computation is done in parallel. In conclusion, HEADSS offers orders

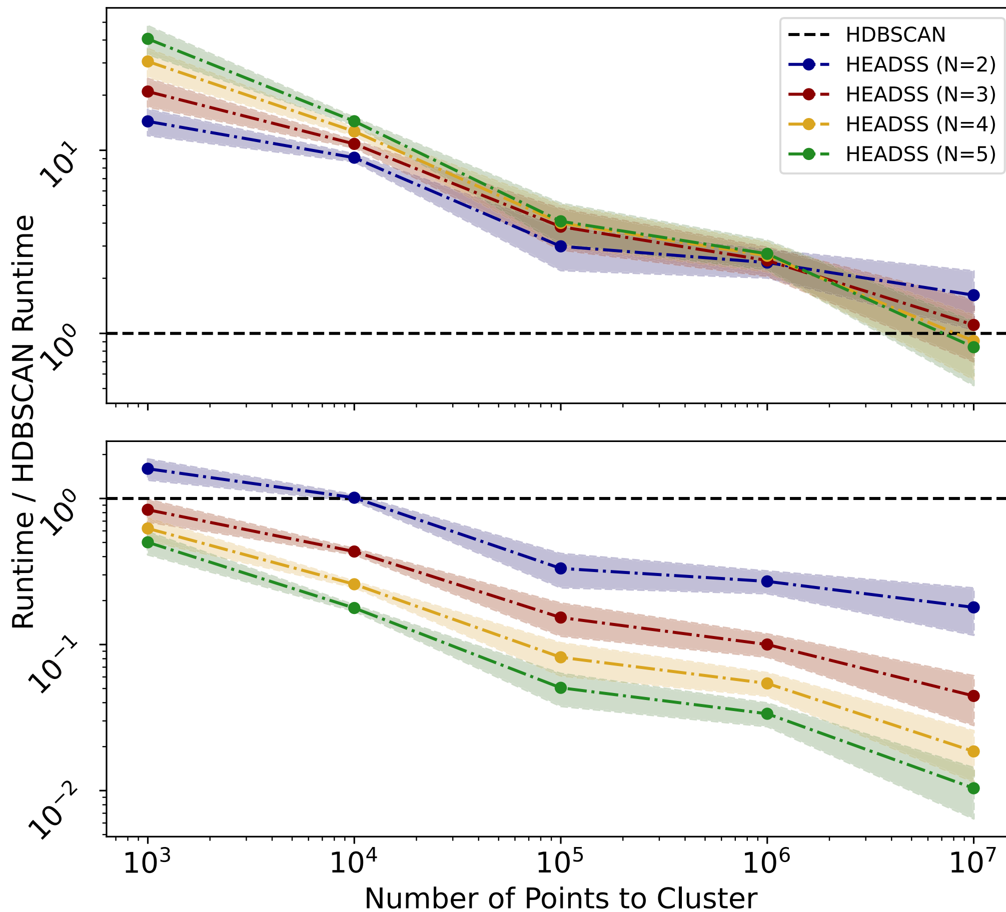


Fig. 8. *Top*: Non-distributed runtimes for HEADSS with multiple N values compared to the standard HDBSCAN implementation for a 2−dimensional feature space with an increasingly large catalogue. *Bottom*: Theoretical distributed runtime for HEADSS (runtimes from the top plot are scaled by a factor of N^D), assumes a fully concurrent implementation and evenly distributed dataset.

of magnitude improvement in theoretical runtimes with as little as 10^7 objects with a considerable reduction in peak memory requirements for a single process.

7. Broader impact

Clustering is commonplace within science, with physical interpretations and theories forming from the outcome (Xu and Wunsch, 2010; Dumont et al., 2018; Kounkel and Covey, 2019). This work brings a standard implementation that produces reliable and repeatable results for any clustering algorithm. To our knowledge, this is the first available package that formalises this process. Due to the careful consideration of complexity and memory requirements throughout, we have also ensured scalability by reducing the data size of a given partition and reduced run times even in single-threaded implementations, see Fig. 8. In testing the memory limit for HDBSCAN is reached, whereas, in HEADSS this limit represents the largest partition possible defining the minimum value of N. Furthermore, there is great potential for further improvements in run times through the utilisation of workflow managers such as Slurm (Yoo et al., 2003).

Scaling HEADSS for very large datasets is relatively straightforward. Sufficiently large datasets can be split across a proportionally large number of partitions, reducing the peak computational requirements to sensible levels. In practice, the optimal N used for splits depends on many factors including the composition of features, number of compute nodes, size of compute nodes and the importance of runtime. In practice, a large dataset of 10^{12} entries with a fixed $N = 5$ split across two features results in 81 partitions, each with $\sim 10^{10}$ entries. This scale remains a considerable task, hence splitting over three or four features results in 729 or 6561 partitions respectively. Such a split over four features has reduced our peak requirements from clustering one trillion data points to roughly 200–300 million. Alternatively, if only two suitable splitting features exist, we could partition the data with $N = 16$, creating 961 partitions, each with approximately 1 billion data points. The only caveat to these extremes is that the largest scale structure is likely to be lost, although it is likely possible to identify this structure using lower-resolution data. A good example would be using maps of stellar positions stars to identify galaxy filaments. In practice, the galaxy positions would be more suitable and represent a much smaller catalogue.

Returning to the challenge that inspired this work, identifying stellar clusters and co-moving groups within the Milky Way plane. HEADSS offers an opportunity to influence clustering with prior domain and scientific knowledge. The clustering of stellar clusters requires both positional (coordinates) and velocity data (proper motions). Nonetheless, there is a fundamental understanding that the positional association outweighs the velocity connections, i.e. objects moving in the same direction but scattered across the Milky Way are not part of a common stellar cluster. The solution offered by HEADSS is splitting the data by the positional data and continuing to cluster using both data types does not negatively affect the clusters identified. Essentially, the proper motion data increases the accuracy, but as a secondary subset of features for this dataset have a lower feature importance. Clustering the Milky Way plane offers an excellent use

case for HEADSS due to the natural feature importance hierarchy and natural distribution across the positional feature space. The results, to be published as part of Crake et al. (In Prep), represent a true scientific use case with a dataset of $\sim 10^8$ objects. For use in other datasets, a general rule is to partition the data along features with the highest importance.

8. Conclusions

This work presents HEADSS, a package designed to facilitate the accurate clustering of big data. We aimed to make any clustering algorithm scalable and repeatable while retaining its accuracy and reducing the size of required compute nodes. We have shown HEADSS represents an improvement in both scalability and run times, using HDBSCAN as an example, across a range of datasets representing a plethora of potential use cases. We also provide a process of merging clusters to expand the potential impact of this work way beyond the specific use cases first envisioned. We do not explore the limitations of HEADSS for high dimensionality as the process is optimised for splitting along minimal features whilst allowing the selected algorithm to cluster on additional features.

CRediT authorship contribution statement

D.A. Crake: Software, Methodology, Project idealization, Visualization, Writing – original draft. **N.C. Hambly:** Supervision, Writing – review & editing. **R.G. Mann:** Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: We would like to thank the referees for their suggestions and comments, as well as Dave Morris and Stelios Voutsinas for their assistance with implementing this software on the Gaia Data Mining Platform. This work was funded by UKRI through grant ST/S0019481 (NCH) and by the award of a PhD studentship to DAC, while RGM acknowledges the University of Edinburgh for their support. For the purpose of open access, the authors have applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising from this submission

Data availability

Data will be made available on request

References

Ajin, V.W., Kumar, L.D., 2016. Big data and clustering algorithms. In: 2016 International Conference on Research Advances in Integrated Navigation Systems. RAINS, pp. 1–5. doi:10.1109/RAINS.2016.7764405.

Ali, T., Asghar, S., Sajid, N.A., 2010. Critical analysis of DBSCAN variations. In: 2010 International Conference on Information and Emerging Technologies. pp. 1–6. doi:10.1109/ICIET.2010.5625720.

Balcan, M.F., Ehrlich, S., Liang, Y., 2013. Distributed K-means and k-median clustering on general topologies. In: Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2. NIPS '13, Curran Associates Inc., Red Hook, NY, USA, pp. 1995–2003.

Bateni, M., Behnezhad, S., Derakhshan, M., Hajiaghayi, M., Kiveris, R., Lattanzi, S., Mirrokni, V., 2017. Affinity clustering: Hierarchical clustering at scale. Adv. Neural Inf. Process. Syst. 30.

Chang, H., Yeung, D.Y., 2008. Robust path-based spectral clustering. Pattern Recognit. 41, 191–203. doi:10.1016/j.patcog.2007.04.010.

Crake, D.A., Mann, R.G., Hambly, N.C., 2023. TBA (in preparation) Unpublished Manuscript.

Dumont, M., Reninger, P., Pryet, A., Martelet, G., Aunay, B., Join, J., 2018. Agglomerative hierarchical clustering of airborne electromagnetic data for multi-scale geological studies. J. Appl. Geophys. 157, 1–9. doi:10.1016/j.jappgeo.2018.06.020, URL: https://www.sciencedirect.com/science/article/pii/S0926985117301891.

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining. KDD '96, AAAI Press, pp. 226–231.

Fränti, P., Sieranoja, S., 2018. K-means properties on six clustering benchmark datasets 48 (12). pp. 4743–4759, URL: http://cs.uef.fi/sipu/datasets/.

Fu, L., Medico, E., 2007. FLAME, a novel fuzzy clustering method for the analysis of DNA microarray data. BMC Bioinformatics 8, 3. doi:10.1186/1471-2105-8-3.

Gionis, A., Mannila, H., Tsaparas, P., 2007. Clustering aggregation. ACM Trans. Knowl. Discov. Data (TKDD) 1, 1–30.

Jain, A.K., Law, M.H.C., 2005. Data clustering: A user's dilemma. In: Pal, S.K., Bandyopadhyay, S., Biswas, S. (Eds.), Pattern Recognition and Machine Intelligence. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 1–10.

Kärkkäinen, I., Fränti, P., 2002. Dynamic Local Search Algorithm for the Clustering Problem. Technical Report A-2002-6, Department of Computer Science, University of Joensuu, Joensuu, Finland.

Karypis, G., Han, E.-H., Kumar, V., 1999. Chameleon: Hierarchical clustering using dynamic modeling. Computer 32 (8), 68–75.

Kounkel, M., Covey, K., 2019. Untangling the Galaxy. I. Local Structure and Star Formation History of the Milky Way. Astron. J. 158 (3), 122. doi:10.3847/1538-3881/ab339a, arXiv:1907.07709.

Kumar, A., Ingle, Y.S., Pande, A., Dhule, P., 2014. Canopy clustering: a review on pre-clustering approach to K-means clustering. Int. J. Innov. Adv. Comput. Sci.(IJIACS) 3 (5), 22–29.

Lattanzi, S., Lavastida, T., Lu, K., Moseley, B., 2020. A framework for parallelizing hierarchical clustering methods. In: Brefeld, U., Fromont, E., Hotho, A., Knobbe, A., Maathuis, M., Robardet, C. (Eds.), Machine Learning and Knowledge Discovery in Databases. Springer International Publishing, Cham, pp. 73–89.

Lucic, M., Bachem, O., Krause, A., 2016. Strong coresets for hard and soft bregman clustering with applications to exponential family mixtures. In: Gretton, A., Robert, C.C. (Eds.), Proceedings of the 19th International Conference on Artificial Intelligence and Statistics. In: Proceedings of Machine Learning Research, vol. 51, PMLR, Cadiz, Spain, pp. 1–9, URL: https://proceedings.mlr.press/v51/lucic16.html.

McCallum, A., Nigam, K., Ungar, L.H., 2000. Efficient clustering of high-dimensional data sets with application to reference matching. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 169–178.

McInnes, L., Healy, J., Astels, S., 2017. Hdbscan: Hierarchical density based clustering. J. Open Source Softw. 2 (11), doi:10.21105/joss.00205.

Michelson, M., Knoblock, C.A., 2006. Learning blocking schemes for record linkage. In: AAAI, Vol. 6. pp. 440–445.

Moon, T., 1996. The expectation-maximization algorithm. IEEE Signal Process. Mag. 13 (6), 47–60. doi:10.1109/79.543975.

Na, S., Xumin, L., Yong, G., 2010. Research on k-means clustering algorithm: An improved k-means clustering algorithm. In: 2010 Third International Symposium on Intelligent Information Technology and Security Informatics. pp. 63–67. doi:10.1109/IITSI.2010.74.

O'Hare, K., Jurek-Loughrey, A., de Campos, C., 2019. An unsupervised blocking technique for more efficient record linkage. Data Knowl. Eng. 122, 181–195. doi:10.1016/j.datak.2019.06.005, URL: https://www.sciencedirect.com/science/article/pii/S0169023X18306098.

Sagheer, N.S., Yousif, S.A., 2021. Canopy with k-means clustering algorithm for big data analytics. In: American Institute of Physics Conference Series, Vol. 2334. 2334, 070006. doi:10.1063/5.0042398.

Sieranoja, S., Fränti, P., 2019. Fast and general density peaks clustering. Pattern Recognit. Lett. 128, doi:10.1016/j.patrec.2019.10.019.

Singh, K., Malik, D., Sharma, N., 2011. Evolving limitations in K-means algorithm in data mining and their removal. IJCEM Int. J. Comput. Eng. Manag. 12, 2230–7893, URL: http://ijcem.org/papers42011/42011_26.pdf.

Sumengen, B., Rajagopalan, A., Citovsky, G., Simcha, D., Bachem, O., Mitra, P., Blasiak, S., Liang, M., Kumar, S., 2021. Scaling hierarchical agglomerative clustering to billion-sized datasets. arXiv e-Prints https://doi.org/10.48550/arXiv.2105.11653.

Veenman, C., Reinders, M., Backer, E., 2002. A maximum variance cluster algorithm. IEEE Trans. Pattern Anal. Mach. Intell. 24, 1273–1280. doi:10.1109/TPAMI.2002.1033218.

Xu, R., Wunsch, D.C., 2010. Clustering algorithms in biomedical research: a review. IEEE Rev. Biomed. Eng. 3, 120–154.

Yoo, A.B., Jette, M.A., Grondona, M., 2003. SLURM: Simple linux utility for resource management. In: Feitelson, D., Rudolph, L., Schwiegelshohn, U. (Eds.), Job Scheduling Strategies for Parallel Processing. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 44–60.

Zhang, Y., Zhao, Y., 2015. Astronomy in the big data era. Data Sci. J. 14, 11. doi:10.5334/dsj-2015-011, URL: https://datascience.codata.org/articles/10.5334/dsj-2015-011/#.