INCREMENTAL NON-GREEDY CLUSTERING AT SCALE

A Dissertation Presented

by

NICHOLAS MONATH

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

February 2022

Manning College of Information and Computer Sciences

INCREMENTAL NON-GREEDY CLUSTERING AT SCALE

A Dissertation Presented

by

NICHOLAS MONATH

Approved as to style and content by:

Andrew McCallum, Chair

Daniel Sheldon, Member

Marco Serafini, Member

Christos Faloutsos, Member

Alexander J. Smola, Member

James Allan, Chair of the Faculty
Manning College of Information and Computer Sciences

DEDICATION

To my parents.

ACKNOWLEDGMENTS

I would like to extend my deepest thanks and gratitude to my advisor, Andrew McCallum, for his guidance, mentorship, support and caring leadership during my PhD. As my advisor, he helped me to become a better researcher, as well as a better collaborator, friend, and person. He made my time at UMass a truly meaningful and special experience. I've learned many life lessons from him through both his words and actions. Andrew, you are so supportive and compassionate towards others; treat everyone with respect; and I appreciate how lift up your students to do their best work and be their best selves. Thank you for your patience, support and advising; they have been invaluable to me and allowed me to grow as a researcher. I am so grateful to have been taken under your wing and introduced by you to the world of research and machine learning.

Thank you to my committee members: Daniel Sheldon, Marco Serafini, Alex Smola, and Christos Faloutsos. I am grateful for your careful considerations of my work, your time, as well as the unique perspectives you each gave me. Dan, thank you for your thoughtful questions and clear insights on my proposal and defense. Marco, thank you for raising interesting questions related to scalability, systems, and bottlenecks. Christos, thank you so much for meeting and advising me many times through this process. I appreciate your caring and thoughtful perspectives on this work and your careful reading and suggestions. Alex, thank you for your astute questions and thoughtful perspectives on future work and how to focus my research in ways to achieve broader applicability.

In the first several years of my PhD, I had the fortuitous opportunity to have Ari Kobren as my closest collaborator. Working with you, Ari, was not only a technical delight, but you also made it so much fun. Observing the way Ari approaches work–building technical depth, brainstorming wild ideas, and carefully analyzing and thoughtfully presenting results– was

inspiring and a fantastic collaboration experience. I loved our pair programming sessions in IESL, walks to discuss results, late night deadlines and washing dishes in the CICS kitchen. From the ways to write research code, good papers, and clear presentations to making Kombucha, I've learned so much from you and I am grateful to be your friend and collaborator. Wishing you, Shilpa, Tal, and Rami all the best in the years to come.

I am also grateful for the friendship of Rajarshi Das. The PhD process has been quite a journey together. I can't overstate how meaningful it has been to me to have you in my life and share these adventures and challenging times. You have been supportive in hardships. I'll always remember constantly laughing about random things, all the coffee in the lab, the Yoda mug, going running, dinners at Fresh Side with all the guest speakers, and just how caring and thoughtful you are to all your friends and those around you. As a researcher, I'm inspired by your years-forward looking perspectives and tenacity.

By way of Rajarshi, I am grateful to have met Manzil Zaheer. Manzil, I am tremendously appreciative of your influence and training to improve my research skills, critical thinking, and creativity. I give my heartfelt thanks to you, Manzil, for countless hours of help, mentorship, guidance, and feedback. Being your intern mentee was so much fun and I am grateful for having gained such an amazing friend. Great times included: relating clustering CovType to Eddington's solar eclipse experiment; you showing me nearest neighbor search proofs in Bay Area restaurants; and not to mention all the parsnips and pistachio ice cream. And I truly appreciate all of your friendship and support in the pandemic (esp. the early days), virtual hangouts, and discussing big ideas. I am grateful for all of your help with my thesis proposal and defense as well as all the puzzles you share.

I am also grateful to all of the collaborators and people who have helped me during my PhD. Akshay Krishnamurthy, your guidance and mentorship in the Perch and Grinch papers was transformational and I am grateful to have had your help with the breakthroughs that made those papers possible. Luke Vilnis, maybe it is no surprise to mention you here as the most acknowledged member of IESL, but your pointers and suggestions were also crucial

for that work and helpful to me. Thanks for insights and friendship. Thank you also to my fantastic collaborators: Pat Flaherty, Kyle Cranmer, Mrinmaya Sachin, and Cameron Musco.

I am extremely grateful for my internship mentor, Amr Ahmed, who welcomed me into his group and inspired me with new perspectives and challenging research problems. I am grateful that he included me in research projects in his group and for his enthusiasm and drive for publishing. I am moved by his support and tireless work provide a meaningful internship experience and to introduce new approaches and tasks into our work together. I also give my sincerest thanks to Avinava Dubey as well for making my intern experience so much fun and being a great mentor and collaborator. Avi, working with you to break down a research problem is such an enjoyable collaborative experience, even without mango and cappuccinos. Your energy and many-horse-power enthusiasm for coming up with new ideas is contagious and inspiring. I would also like to thank Guru Guruganesh for all his help and discussions in my internships. The clarity with which Guru approaches research is inspiring. He asks such astute questions and offered so many helpful insights and perspectives. I would like to express my gratitude to Gokhan Mergen and Mert Terzihan for welcoming me and including me in their research projects. I would like to thank Dan Silva for his superb figure making and data analysis help in our papers together. I would like to thank Yuan Wang for his assistance with experiments as well. Finally, I would like to thank Marc Najork and Aranyak Mehta for welcoming me into their teams. Thank you all for making those winter internships a fantastic experience.

I would also like to thank the other superb internship mentors that I worked with: Shankar Ananthakrishnan, Mukund Harakere Sridhar, Bo Xiao - your mentorship was formative in my PhD journey; Michael Glass, Gaetano Rossiello, Robert Farrell, Sarthak Dash, Nicolas Fauceglia, and Alfio Gliozzo - you inspired me to think of new connections between research areas; and Ivana Williams, Sunil Mohan, E.C. Caley, Sam Molyneux - I truly appreciate your enthusiasm in our collaboration, inspiring me with problems whose solutions impact

your end-users, and further instilling in me the values and importance of diving deeply into data. I would also like to thank my collaborators at PatentsView, Christina Jones, Sarvo Madhavan, Chris DiPietro, Evgeny Klochikhin, and Andy Toole. Thank you for all our work together over these years. I truly appreciate you giving me the opportunity to work directly with you on PatentsView entity resolution.

I want to thank all of the friends that I was so fortunate to meet while at UMass: Craig Greenberg - for his humor and selfless collaboration; Ian Gemp - for his sound advice; Arvind Neelakantan - for his kindness; Archan Ray - for his consistent excitement; Jinho Choi - for first introducing me to Andrew and inspiring me to pursue research in his class; David Belanger, Luke Vilnis, Emma Strubell, Pat Verga, Ben Roth - for their mentorship; Javier Burroni - for always lending a thoughtful perspective and for his friendship and support; Michael Boratko - for being my pandemic buddy and such a thoughtful collaborator and mentor; Trapit Bansal and Haw-Shiuan Chang - for all of their help, thoughtful comments, and support over the years; Nishant Yadav and Rico Angell - for their collaboration and including me in their work; Tomas Geffner - for his friendship and dining suggestions in NYC; Andrew Drozdov - for his help with Data Science Tea; Dongxu Zhang - for also being my pandemic buddy and for our collaborations; Aaron Traylor, Derek Tam, Raghuveer Thirukovalluru, Dhruv Agarwal, Sid Mishra, Johnny Wei, and others - for graciously being my mentees and teaching me so much; Aaron Schein, Abe Handler, Adam Saunders, Ajay Nagesh, Ameya Godbole, Anupama Pasumarthy, Aruni RoyChowdhury, Caiti Cellier, Dan Barowy, Dhruvesh Patel, Dung Thai, Francisco Garcia, Garrett Bernstein, Jack Sullivan, Jay Yoon Lee, Jeevan Shankar, Jeffrey Flanigan, Justin Payan, Kate Silverstein, Klim Zaporojets, Lakshmi Vikraman, Melisa Bok, Michael Spector, Mohit Yadav, Mrinal Das, Neha Kennard, Oskar Singer, Pedram Rooshenas, Sachin Bhat, Shehzaad Dhuliawala, Sheshera Mysore, Shib Sankar Dasgupta, Shikhar Murty, Subendhu Rongali, Tim O'Gorman, Xiang Lorraine Li, - for making IESL and UMass such a great place to me. IESL was a family for me and I could not think of a better group of people to work with. Thank you all!

I am grateful for the UMass staff for all that they do to make the PhD process possible: Pam Mandler, Lynn Yovina, Elena Hayes, Brant Cheikes - for all of their help, especially organizing the Data Science Tea; Yvonne Crevier, Glenn Stowell, Marla Michel - for all their help with grants, assistantships, and external collaborations; Dan Parker - for his dedication to making such a fantastic computing environment; Eileen Hamel, Leeanne Leclerc, Malaika Ross, and Kyle Skemer - for their support in the CICS program; and Barb Sutherland for always being a warm and helpful presence in the CICS office.

I am also grateful to my undergraduate advisors, Antonella Di Lillo and Jim Storer for inspiring me to pursue this degree. Thank you also to Mykel Kochenderfer for his mentorship and inspiration during my undergraduate years.

Thank you to Asher for his consistent friendship and support over all these years and for always being there to talk me through good and bad times. You are a true friend. Thank you to Christian for his lifetime of friendship and support and for always helping me find the way. Thank you to Mary Blake for her friendship and support from the days when we would walk to town to the present where she held me accountable on this document with our weekly stand-ups. Thank you to Tommy for his unconditional friendship and support as well as his philosophical perspectives. Thanks to Chris for his friendship and all the music. Thank you to all Brandeis friends, especially to Jordan Pardo and David Lasher for our first research project. Thank you to Katherine for her love, support and patience through the years and for always understanding when I had a paper deadline.

Finally, thank you to my parents, Blake and Tom, for all of their love, guidance, sacrifices, and support to help me in life and in graduate school. I will always be grateful to you for providing me with opportunities, helping me through so many things, and always being there for me.

ABSTRACT

INCREMENTAL NON-GREEDY CLUSTERING AT SCALE

FEBRUARY 2022

NICHOLAS MONATH

B.Sc., BRANDEIS UNIVERSITY

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Andrew McCallum

Clustering is the task of organizing data into meaningful groups. Modern clustering applications such as entity resolution put several demands on clustering algorithms: (1) scalability to massive numbers of points as well as clusters, (2) incremental additions of data, (3) support for any user-specified similarity functions.

Hierarchical clusterings are often desired as they represent multiple alternative flat clusterings (e.g., at different granularity levels). These tree-structured clusterings provide for both fine-grained clusters as well as uncertainty in the presence of newly arriving data. Previous work on hierarchical clustering does not fully address all three of the aforementioned desiderata. Work on incremental hierarchical clustering often makes greedy, irrevocable clustering decisions that are regretted in the presence of future data. Work on scalable hierarchical clustering does not support incremental additions or deletions. These methods often make requirements on the similarity functions used and/or empirically tend to over merge clusters, which can lead to inaccurate clusterings.

In this thesis, we present incremental and scalable methods for hierarchical clustering to empirically satisfy the above desiderata. Our work aims to represent uncertainty and meaningful alternative clusterings, to efficiently reconsider past decisions in the incremental case, and to use parallelism to scale to massive datasets. Our method, GRINCH, handles incrementally arriving data in a non-greedy fashion, by reconsidering past decisions using tree structure re-arrangements (e.g., rotations and grafts) invoked in accordance with the user's specified similarity function. To achieve scalability to massive datasets, our method, SCC, builds a hierarchical clusterings in a level-wise bottom-up manner. Certain clustering decisions are made independently in parallel within each level, and a global similarity threshold schedule prevents greedy over-merging. We show how SCC can be combined with the tree-structure re-arrangements in GRINCH to form a mini-batch algorithm achieving both scalable and incremental performance. Lastly, we generalize our hierarchical clustering approaches to DAG-structured ones, which can better represent uncertainty in clustering by representing overlapping clusters. We introduce an efficient bottom-up method for DAG-structured clustering, LLAMA. For each of the proposed methods, we provide both a theoretical and empirical analysis. Empirically, our methods achieve state-of-the-art results on clustering benchmarks in both the batch and the incremental settings, including multiple point improvements in dendrogram purity and scalability to billions of points.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

Clustering, the partitioning of points into disjoint sets, is an extensively used tool in data science and is central to unsupervised machine learning. Clustering is used by practitioners for analyzing and visualizing large datasets [58, 85, 164, 178, 239, 257, 267, 276, 304, 306, inter alia]. For instance, it is used to facilitate the exploration of cellular diversity in single-cell transcriptomics data [245] and to understand the behavioral patterns of users of social networks [32]. Clustering is also used to solve tasks such as entity resolution, in which ambiguous mentions of entities are clustered together such that each cluster represents a real world entity [42, 89, 104, 180, 183, 184, 192, 237, 258, 259, inter alia]. This task (which is closely related to record linkage, de-duplication, and coreference) is used to build knowledge-bases of scientists from citation records [89, 283, 303], product catalogues [269], and databases regarding human rights data from the conflict in Syria [73]. Furthermore, clustering is widely used in feature extraction [54] and as a building block of larger models, e.g., to identify semantically related sentences for computational-based debating system [107] and to discover latent entity types in knowledge base link prediction [92]. Clustering is also used in recommendation [302] as well as in tasks such as image segmentation [189]. While it is the case that many clustering tasks are NP-hard [93, 94, 196] and impossible to satisfy three simple properties (scale-invariance, a richness, consistency) [167], clustering is widely used and beneficial to the aforementioned applications in practice.

Clustering is commonly categorized into two families: the aforementioned *flat* and the tree-structured, nested partitions of data, *hierarchical*. In a hierarchical clustering, the leaves correspond to data points and the internal nodes correspond to clusters of their descendant

leaves. The nested clustering structure of the hierarchies can be useful to represent clusters of multiple granularity [302] or to automatically discover structures including as word or concept ontologies [44, 206, 207, 231, 280, 293]. Hierarchical clustering methods are used to discover phylogenetic trees [122, 212, 233, 279] as well as to discover the structure of particles in jet physics [87, 134].

With each internal node of a hierarchical clustering representing a cluster, collections of internal nodes can represent a flat clustering, known as a *tree consistent partitions* [146]. This illustrates how hierarchical clusterings can be used to represent uncertainty about a candidate flat clustering. Indeed, flat partitions are often selected after first constructing a hierarchical clustering. Empirically, this has been show to be effective in entity resolution [131, 183, 303]. Theoretically, this has been useful for K-Means clustering [136]. Interactive methods can also be used to incorporate user feedback when selecting a flat clustering [171, 275].

Hierarchical agglomerative clustering (HAC) [18, 95, 96, 129, 217, 218, 241, 253, 281, inter alia], the best-first, bottom-up algorithm, is one of the most widely-used clustering algorithms [44, 89, 108, 131, 172, 180, 245]. It is used as the basis for inference in many statistical models [44, 45, 145, 146], as an approximation algorithm for hierarchical [94, 214] and flat [136] clustering costs as well as for supervised clustering [163, 288]. One capability that contributes to HAC's prevalence is that it can be used to construct a clustering according to any, including a learned, cluster-level scoring function, also known as a *linkage function* [81, 89, 163, 182, 288].

There are two key limitations of HAC: (1) it does not support incremental additions and deletions of data and (2) it has limited scalability in terms of dataset size. In many applications, data is continuously arriving over time. Newly arrived data points need to be incorporated into the predicted flat/hierarchical clusterings. These data points might correspond to new clusters / branches of the hierarchical clustering tree or may be added to existing clusters. The addition of new points to the clustering must be done efficiently, without recomputing a clustering over the entire dataset.

Much of the previous work on incremental / online clustering focuses heavily on efficiency while making irrevocable, greedy decisions that can lead to inaccurate clusterings [57, 66, 114, 300]. In our work, we use hierarchical clustering to represent uncertainty in this online setting [170, 210]. Our methods add newly arriving data points to a region of the tree containing the points' nearest neighbor (according to the given similarity measure). Then in a non-greedy fashion, tree structure re-arrangements are made to efficiently and effectively reconsider past decisions. Specifically, a rotation operation re-arranges local structure and a grafting operation considers global re-arrangements. These operations allow our methods to be robust even when data arrives in adversarial orderings.

While online/incremental methods have efficient per-point addition times, the sequential nature of these methods can only mildly use parallelism. Affinity clustering [31], overcomes the main computational expense of HAC by using a bottom-up algorithm merging connected components of the 1-nearest neighbor graph. Affinity clustering can use efficient parallel and distributed connected component algorithms [165, 301] as well as graph contraction [31]. While efficient, Affinity clustering can empirically suffer from over-merging clusters [209]. We propose a closely related hierarchical clustering algorithm, the *Sub-Cluster Component Algorithm (SCC)*, which can effectively interpolate between the speed of affinity and the accuracy of HAC. Like Affinity clustering, SCC builds tree structures one level at a time with clustering decisions independently in parallel merging sub-clusters. Unlike Affinity clustering, we use a sequence of level-wise thresholds to determine in which level certain mergers of sub-clusters should take place, thereby preventing over-merging and improving the accuracy of the predicted clustering.

SCC and Affinity do not support incremental additions of points. We extend SCC to a mini-batch incremental algorithm, MBSCC, which combines the tree re-arrangements of GRINCH and the level-wise structure of SCC. Rotation operations select which level of the tree structure the new points should be added while grafts allow previously constructed clusters in a level to be split or merged in the presence of new data. The re-arrangements

provide for an efficient algorithm that only needs to reconsider a small portion of the tree structure in the presence of new data.

One limitation of hierarchical clustering is the kind of overlapping clusters that it can represent [59, 88, 120, 188]. In particular, all overlapping clusters must exhibit a nested relationship. This disallows leaf data points to simultaneously sit in multiple overlapping, but non-nested, clusters and requires that leaves and internal nodes have a single parent in the structure. Removing the tree-based constraints, we can instead aim to discover *DAG-structured* clusterings. Jeantet et al [156] provide a bottom-up approach for building DAGs, but this is limited to datasets of a thousands of points. In our work, we present a scalable algorithm, LLAMA, for discovering meaningful DAG-structured clusterings on large datasets using a level-wise bottom-up approach.

For each of the proposed methods, we provide both a theoretical and empirical analysis. Our theoretical analysis describes each method in terms of a model-based separation condition, which generalizes popular data assumptions including strict separation [26] and delta separation [177]. Our empirical analysis considers performance of our methods in batch settings, online/incremental settings, adversarial data orderings, standard/learned similarities on both clustering and entity resolution benchmarks. Our methods achieve multiple point improvements in dendrogram purity and scalability to billions of points.

1.1 Summary of Desiderata

Our work aims to develop clustering algorithms that have the following properties:

Scale to many clusters, not just many points Clustering methods should accurately discover a large number of clusters from a large number of points. These methods are needed for tasks like entity resolution and to discover meaningful fine-grained clusters.

Clusters of multiple granularity Algorithms should discover overlapping, nested clusters that capture uncertainty in the clustering as well as inherent ambiguity in the data.

Incremental addition of data We need clustering algorithms that can effectively incorporate and adapt to a continuous stream of arriving data. The algorithms should support the creation of new structure and effectively and efficiently reconsider past decisions.

Non-greedy decisions A core difficultly in handling incremental data is the ability to efficiently reconsider past clustering decisions. This allows these algorithms to be non-greedy in the presence of newly arriving data.

Support arbitrary similarity measures We would like our proposed approaches to handle any user specified similarity measure.

Guarantees and Analysis We would like to provide theoretical statements to ensure our proposed methodology is effective when data meets certain straightforward clustering properties.

1.2 Summary of Work

The core methodologies proposed in this thesis were first described in:

- Ari Kobren*, Nicholas Monath*, Akshay Krishnamurthy, and Andrew McCallum. *A Hierarchical Algorithm for Extreme Clustering.* KDD. 2017.
- Nicholas Monath*, Ari Kobren*, Akshay Krishnamurthy, Michael Glass, Andrew McCallum. *Scalable Hierarchical Clustering via Tree Grafting*. KDD. 2019
- Ari Kobren, Nicholas Monath, Andrew McCallum. *Integrating User Feedback under Identity Uncertainty in Knowledge Base Construction*. AKBC, 2019.

- Nicholas Monath, Avinava Dubey, Guru Guruganesh, Manzil Zaheer, Amr Ahmed, Andrew McCallum, Gokhan Mergen, Marc Najork, Mert Terzihan, Bryon Tjanaka, Yuan Wang, Yuchen Wu. *Scalable Hierarchical Agglomerative Clustering* . KDD, 2021.

- Nicholas Monath, Manzil Zaheer, Avinava Dubey, Amr Ahmed, Andrew McCallum. *DAG-structured Clustering by Nearest-Neighbors*. AISTATS 2021.

This work includes:

Incremental Non-Greedy Hierarchical Clustering Our method GRINCH (Chapter 3) supports the incremental additions and deletions of data points. It uses a hierarchy to represent multiple alternative clusterings of a dataset and tree re-arrangements to effectively and efficiently reconsider past decisions. GRINCH supports any similarity measure / linkage function and generalizes the PERCH algorithm which is specialized for Euclidean space [170].

Level-wise Hierarchical Clustering Our method SCC (Chapter 4) builds tree structures one level at a time with clustering decisions made independently in parallel within each level. This provides for greater scalability than the sequential clustering of GRINCH. To support incremental addition of points, we present Mini-batch SCC (MBSCC), which efficiently uses the level-wise structure of SCC to parallelize across points in a mini-batch. Like GRINCH, MBSCC uses tree re-arrangements to support non-greedy decisions in the presence of incrementally arriving data.

DAG-Structured Clustering Rather than constructing a tree structure, our method LLAMA (Chapter 5) builds a DAG-structure of nested clusters in a bottom-up, level-wise manner. The DAG-structure allows us to discover overlapping, but not nested clusterings that can represent both uncertainty and ambiguity in the data.

1.3 Document Organization

This document is organized as follows: Chapter 2 provides definitions of the clustering structures and assumptions as well as background algorithms and related work; Chapter 3 describes our algorithm GRINCH for incremental hierarchical clustering; Chapter 4 describes the level-wise, parallelizable algorithms and their incremental extensions; and Chapter 5 describes our work on DAG-structured clustering. Experimental and theoretical results for each method are given within each of the corresponding chapters.

CHAPTER 2

BACKGROUND

In this section, we formally define the clustering problems studied in this thesis, introducing the notation that is used throughout. We define the clustering evaluation measures that are used. We also provide a description of the data separability assumptions used in the theoretical analysis with examples of generative models of data that satisfy them. We provide an overview of related work.

2.1 Problem Definition

We are given a dataset $\boldsymbol{X} = \{x_1, x_2, \ldots, x_N\}$ and a set-wise similarity function (Section 2.3), $f : \mathcal{P}(\boldsymbol{X}) \times \mathcal{P}(\boldsymbol{X}) \to \mathbb{R}$. Our goal is to discover meaningful flat (Definition 2), hierarchical (Definition 3), and DAG-structured (Definition 5) clusterings. We aim to discover clusterings which are well aligned with a ground truth clustering (Section 2.4). Rather than optimizing a cost function, we aim to design algorithms that recover data that is *model-based separated* (Section 2.5.1).

2.2 Definitions

We use $\boldsymbol{X}$ to refer to a dataset of points $\boldsymbol{X} = \{x_1, x_2, \ldots, x_N\}$. We use $\mathcal{P}(\boldsymbol{X})$ to refer to the power set of $\boldsymbol{X}$, the set of all subsets of $\boldsymbol{X}$. We are interested in several clustering structures corresponding to groupings of elements of $\boldsymbol{X}$. Each of these clustering structures can be described as a set of sets of elements of $\boldsymbol{X}$ as well as in terms of data structure-like definitions. First, consider a set cover, a set of sets such that each element appears in at least one set:

Definition 1. (Set Cover). *A set cover of $\boldsymbol{X}$, denoted $\mathcal{S} \subset \mathcal{P}(\boldsymbol{X}) = \{C_1, \ldots, C_K\}$, is a set of non-empty subsets, $C_i \neq \emptyset$, such that each point in $\boldsymbol{X}$ appears in some subset, $\bigcup_{i=0}^{K} C_i = \boldsymbol{X}$.*

By restricting to disjoint sets, we have the standard definition of a clustering or partition:

Definition 2. (Flat Clustering). *A flat clustering, or partition, of $\boldsymbol{X}$, denoted $\mathcal{C} \subset \mathcal{P}(\boldsymbol{X})$ is a set cover such that the member subsets are disjoint, $\forall C_i, C_j \in \mathcal{C},\ C_i \cap C_j = \emptyset,\ \forall C_i \neq C_j$.*

A hierarchical clustering is a recursive partitioning of dataset into a tree-structured set of nested partitions. Formally:

Definition 3. (Hierarchical Clustering [175]). *A hierarchical clustering, $\mathcal{T}$, of a dataset $\boldsymbol{X} = \{x_1, x_2, \ldots, x_N\}$, is a set of clusters $C_0 = \{x_i\}_{i=1}^{N}$ and for each $C_j, C_k \in \mathcal{T}$ either $C_j \subset C_k$, $C_k \subset C_j$ or $C_j \cap C_k = \emptyset$. For any cluster $C \in \mathcal{T}$, if $\exists C'$ with $C' \subset C$, then there exists a set $\{C_j\}_{j=1}^{\ell}$ of disjoint clusters such that $\bigcup_{j=1}^{\ell} C_j = C$.*

The definition of hierarchical clustering is in terms of subsets of $\boldsymbol{X}$ rather than as a discrete data structure with nodes and edges. There is, however, a direct mapping between the discrete data structure and the set-based definition. In the data structure, the nodes correspond to sets in $\mathcal{T}$. A parent-to-child edge exists between C_{p} and C_{c} if $C_{\mathrm{c}} \subsetneq C_{\mathrm{p}}$ and $\nexists C'$ such that $C_{\mathrm{c}} \subset C' \subset C_{\mathrm{p}}$.

A hierarchical clustering encodes a number of different valid flat clusterings. These flat clusterings are referred to as tree consistent partitions [146]. A tree consistent partition is a set of sub-tree roots in $\mathcal{T}$ that form a flat clustering.

Definition 4. (Tree Consistent Partition [146]) *A tree consistent partition $\mathcal{C}^{(\mathcal{T})}$ for the hierarchical clustering $\mathcal{T}$ of dataset $\boldsymbol{X}$ is a partition of $\boldsymbol{X}$ (Def. 2) and $\mathcal{C}^{(\mathcal{T})} \subset \mathcal{T}$.*

By relaxing the restriction that overlapping clusters in a hierarchical have a nested (sub/super-set relationship), we can consider DAG-structured clusterings:

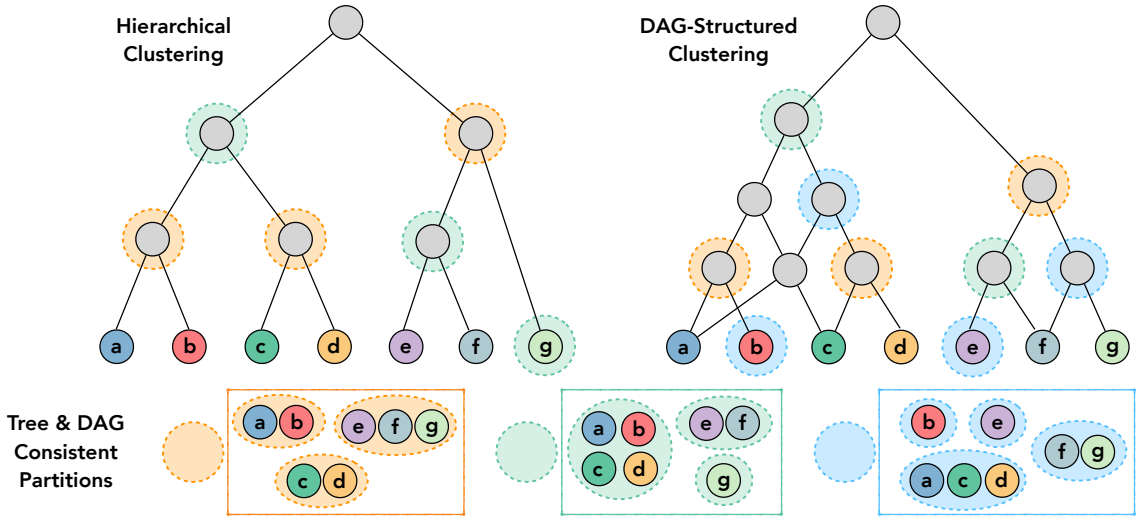


Figure 2.1: **Tree & DAG Consistent Partitions**. The figure shows an example of a hierarchical and DAG-structured clustering of seven points. Three partitions are shown. All three partitions are DAG consistent with the structure on the right. Observe how no tree structure can encode both the green colored partition as well as the blue colored partition since the clusters in the two partitions are overlapping, but not sub/super sets of one another.

Definition 5. (DAG-Structured Clustering) *A DAG-Structured clustering, $\mathcal{D}$, of a dataset $\boldsymbol{X} = \{x_1, x_2, \ldots, x_N\}$, is a subset of the powerset of $\boldsymbol{X}$, $\mathcal{D} \subset \mathcal{P}(\boldsymbol{X})$ containing at least a root node of the entire dataset $\boldsymbol{X} \in \mathcal{D}$ and the shattered partition $\{\{x\} \mid x \in \boldsymbol{X}\} \subset \mathcal{D}$.*

As in the case of tree structures, we can provide a mapping between a DAG data structure and the nested collection of sets. Like the tree structure case, a parent-to-child edge exists between C_{p} and C_{c} if $C_{\mathrm{c}} \subsetneq C_{\mathrm{p}}$ and $\nexists C'$ such that $C_{\mathrm{c}} \subset C' \subset C_{\mathrm{p}}$. In this case, however, a node may have multiple parent nodes, hence calling this a DAG-structured clustering. The set of clusters in a DAG-structured clustering form a partially ordered set, where the containment relationship defines the ordering. The connections to partially ordered sets/Hasse Diagrams have been studied theoretically [59, 88, 120, 188, inter alia]. Like a tree consistent partition, we consider those flat clusterings represented in the DAG structure

Definition 6. (DAG Consistent Partition) *A DAG consistent partition $\mathcal{C}^{(\mathcal{D})}$ for the hierarchical clustering $\mathcal{D}$ of dataset $\boldsymbol{X}$ is a partition of $\boldsymbol{X}$ (Def. 2) and $\mathcal{C}^{(\mathcal{D})} \subset \mathcal{D}$.*

We note that the largest possible DAG-structured clustering contains all elements of the power set of $\mathcal{P}(\boldsymbol{X})$ except the empty set. Of course, such a massive structure is unlikely to be a meaningful clustering and we will design methods that explicitly try to minimize the size of the DAG structure. Figure 2.1 shows examples of tree and DAG consistent clusterings, illustrating a case where DAGs are can represent a richer class of alternative flat clusterings than tree structures.

2.3 Linkage Functions for Clustering

We hope to discover *meaningful* clusterings of data. The definition of *meaningful* will be defined in terms of pairwise similarities between data points and set-wise similarities between clusters. We assume that we are have a *similarity* function between points, $\texttt{sim} : \boldsymbol{X} \times \boldsymbol{X} \to \mathbb{R}$. Further, we assume that we have a similarity defined between two sets of points. Following terminology from HAC, We refer to a similarity function between two sets as a *linkage function*: $f : \mathcal{P}(\boldsymbol{X}) \times \mathcal{P}(\boldsymbol{X}) \to \mathbb{R}$. There are several linkage functions commonly used in HAC such as:

Example 1. (Single Linkage) *Given two sets C_i and C_j, the single linkage is the maximum similarity between an element in C_i and C_j, $f(C_i, C_j) = \max_{x_i, x_j \in C_i \times C_k} \texttt{sim}(x_i, x_j)$.*

Example 2. (Average Linkage) *Given two sets C_i and C_j, the single linkage is the average similarity between pairs of elements in C_i and C_j, $f(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{x_i, x_j \in C_i \times C_k} \texttt{sim}(x_i, x_j)$.*

Example 3. (Complete Linkage) *Given two sets C_i and C_j, the single linkage is the minimum similarity between an element in C_i and C_j, $f(C_i, C_j) = \min_{x_i, x_j \in C_i \times C_k} \texttt{sim}(x_i, x_j)$.*

Unless otherwise noted, we assume that f is a symmetric function, $f(C_i, C_j) = f(C_j, C_i)$ and for notational convenience that the self-similarity of sets is the lowest possible similarity, e.g., $f(C, C) = -\infty$. Linkage functions provide a flexible way to build

and analyze clustering algorithms. Linkage functions may be also learned from data [81, 139, 172, 286, 288, 299].

2.4 Evaluation Measures for Labeled Data

Let's now define how we will measure the quality of flat, hierachical, and DAG-structured clusterings given labeled data. Our evaluation is based on the existence of a ground truth *flat* clustering, $\mathcal{C}^\star$.

2.4.1 Flat Clustering Evaluation Metrics

There are many evaluation metrics used for flat clustering including Adjusted Rand Index [236], normalized mutual information, purity, homogeneity score, and others. We follow work on entity resolution and use the *pairwise F1* metric [89, 283]. Given a dataset $\boldsymbol{X}$ with ground truth clustering $\mathcal{C}^\star$, we measure the pairwise F1 of a predicted clustering $\hat{\mathcal{C}}$. We compute the pairs of points that are in the same cluster in the ground truth:

$$\mathsf{P}^\star = \{(x_i, x_j) \mid \ x_i, x_j \in \boldsymbol{X},\ \exists\, C^* \in \mathcal{C}^\star \text{ s.t. } \{x_i, x_j\} \subseteq C^\star\}, \tag{2.1}$$

and the pairs of points that are in the same cluster in the predicted clustering:

$$\hat{\mathsf{P}} = \{(x_i, x_j) \mid x_i, x_j \in \boldsymbol{X},\ \exists\, C \in \hat{\mathcal{C}} \text{ s.t. } \{x_i, x_j\} \subseteq C\}. \tag{2.2}$$

We then measure precision, recall, and F1:

$$\mathsf{Prec} = \frac{|\mathsf{P}^\star \cap \hat{\mathsf{P}}|}{|\hat{\mathsf{P}}|} \quad \mathsf{Rec} = \frac{|\mathsf{P}^\star \cap \hat{\mathsf{P}}|}{|\mathsf{P}^\star|} \quad \mathsf{F1} = 2 \cdot \frac{\mathsf{Prec} \cdot \mathsf{Rec}}{\mathsf{Prec} + \mathsf{Rec}} \tag{2.3}$$

2.4.2 Hierarchical Clustering Evaluation Metrics

Hierarchical clusterings are often evaluated on datasets for which there exists a ground truth *flat* clustering. One frequently used metric is *dendrogram purity* [146, 151, 170,

263]. Dendrogram purity measures the quality of the flat partitions that are stored in the tree structure without requiring a single partition be selected from the tree structure. Dendrogram purity takes on values between 0 and 1 and achieves a value of 1 if and only if the tree contains the ground truth partition as a tree consistent partition [170]. Given a ground truth flat clustering $\mathcal{C}^\star$ of a dataset X, dendrogram purity of a tree structure $\mathcal{T}$, $\mathsf{DendrogramPurity}(\mathcal{T}, \mathcal{C}^\star)$, as:

$$\mathsf{DendrogramPurity}(\mathcal{T}, \mathcal{C}^\star) = \frac{1}{|\mathsf{P}^\star|} \sum_{C^\star \in \mathcal{C}^\star} \sum_{(x_i, x_j) \in C^\star \times C^\star} \mathsf{purity}(\mathsf{lca}(x_i, x_j, \mathcal{T}), C^\star) \quad (2.4)$$

where $\mathsf{lca}(\cdot, \cdot, \cdot)$ gives the least common ancestor of the nodes in the tree, and $\mathsf{purity}(\hat{C}, C^\star) = \frac{|\hat{C} \cap C^\star|}{|\hat{C}|}$ is the (flat cluster) purity of the cluster represented by $\hat{C}$ with respect to the cluster $C^\star$. In words, dendrogram purity is the average over all pairs of points from the same ground truth cluster of the purity of the pair's least common ancestor.

2.4.3 Metrics for Any Kind of Clustering

To our knowledge, there are not well established metrics to measure the quality of DAG-structured clusterings studied in this thesis. We describe and motivate metrics here that work for flat, hierarchical, and DAG-structured clustering. Further, we note that our metrics can be used in the case that there exists a ground truth *partition* of the data as well as in the case where there is a ground truth *cover* (including tree and DAG structures themselves) of the data (i.e., points are assigned to more than one cluster).

Note that dendrogram purity does not translate well to DAG-structured clusterings. In particular, there exist trivial DAGs that produce perfect dendrogram purity scores (consider the DAG structure containing a set for each pair of points).

2.4.3.1 Recall-Focused Metrics

First, we consider metrics that mimic recall metrics in information retrieval. We measure whether each ground truth cluster is faithfully represented in a tree or DAG-structured

clustering using Jaccard similarity. The Jaccard similarity is defined between a ground truth cluster $C^\star$ and predicted cluster $\hat{C}$: $\textsf{Jacc}(C^\star, \hat{C}) = \frac{|C^\star \cap \hat{C}|}{|C^\star \cup \hat{C}|}$.

Mean Jaccard Per Label We measure the mean over ground truth cluster (or cover) labels of the maximum Jaccard similarity of the ground truth clusters with the predicted structure. This metric was also proposed by [188].

$$\textsf{Jacc/lbl}(\mathcal{D}, \mathcal{C}^\star) = \frac{1}{|\mathcal{C}^\star|} \sum_{C^\star \in \mathcal{C}^\star} \max_{\hat{C} \in \mathcal{D}} \textsf{Jacc}(C^\star, \hat{C}) \tag{2.5}$$

Mean Jaccard Per Point We also compute the mean of the Jaccard similarity over the ground truth points for the highest scoring predicted clustering in the DAG:

$$\textsf{Jacc/pt}(\boldsymbol{X}, \mathcal{D}, \mathcal{C}^\star) = \frac{1}{Z} \sum_{x \in \boldsymbol{X}} \sum_{C^\star \in \mathcal{C}^\star_x} \max_{\hat{C} \in \mathcal{D}} \textsf{Jacc}(C^\star, \hat{C}) \tag{2.6}$$

where $Z = \sum_{x \in X} |\mathcal{C}^\star_x|$ and where $\mathcal{C}^\star_x$ are the ground truth cluster assignments of x. Observe that each metric obtains a value of 1 if and only if each of the ground truth clusters are represented predicted structure.

2.4.3.2 Precision-Focused Metrics

The recall-focused metrics are not enough to measure the quality of a DAG or tree structure. A DAG structure that contains the full powerset $\mathcal{P}(\boldsymbol{X})$ of a dataset would be unmanagably large and contain a multitude of potentially irrelevant substructure. Yet this would achieve a perfect score in terms of the recall-focused metrics. And so, we consider a metric that is focused on the quality of each node in the predicted structures. We encourage structures to be as precise as possible.

Mean Jaccard Per Node We measure mean of the Jaccard similarity of each node with its best aligned ground truth cluster.

$$\textsf{Jacc/node}(\mathcal{D}, \mathcal{C}^\star) = \frac{1}{|\mathcal{D}|} \sum_{\hat{C} \in \mathcal{D}} \max_{\hat{C}^\star \in \mathcal{C}^\star} \textsf{Jacc}(C^\star, \hat{C}) \tag{2.7}$$

2.5 Separability Assumptions

Assumptions are often made about the datasets input to clustering algorithms [21, 26, 28, 170, 177]. Separability assumptions in clustering provide a mechanism to understand whether or not an algorithm effectively and efficiently recovers cluster structure under "reasonable" conditions. While these assumptions might not hold in practice, they represent classes of idealized data that we would like our method to be able to handle. Since the cluster structure is often considered to be 'simpler' when separated than in real datasets of interest to practioners, our goal is to design algorithms that theoretically guarantee performance on this separated data.

We describe our most general classes of separable data, which we call *model-based* separation. We then provide examples of model-based separated data by mapping other, more specific and commonly used separability assumptions to the model-based definition.

2.5.1 Model-based Separation

Model-based separation defines the behavior of a linkage function f with respect to dataset $\boldsymbol{X}$ and a ground truth partition of $\boldsymbol{X}$, denoted $\mathcal{C}^\star$. Model-based separation defines f's behavior in terms of a latent graph structure. This latent graph is unobserved at the time of clustering (i.e., the input to the clustering problem is $\boldsymbol{X}$, not this graph structure). This graph $G = (\boldsymbol{X}, E)$ has one vertex per datapoint in $\boldsymbol{X}$ and edges are defined such that the connected components of G are exactly the clusters of $\mathcal{C}^\star$.

Intuitively, model-based separation says that the linkage function similarity between two sets of points C_0 and C_1 that are directly connected by an edge in G is higher than C_0's or C_1's similarity with any other set C_2 to which it is not directly connected by an edge. Formally, the condition is as follows:

Assumption 1. (Model-based Separation [210]) *Let $G = (\boldsymbol{X}, E)$ be a graph. Let the function $f : \mathcal{P}(\boldsymbol{X}) \times \mathcal{P}(\boldsymbol{X}) \to \mathbb{R}$ be a linkage function that computes the similarity of two groups of vertices and let $\phi : \mathcal{P}(\boldsymbol{X}) \times \mathcal{P}(\boldsymbol{X}) \to \{0, 1\}$ be a function that returns 1 if the*

union of its arguments is a connected subgraph of G. Dataset $\boldsymbol{X}$ is model-based separated with respect to f if:

$$\forall C_0,\ C_1,\ C_2 \subseteq \boldsymbol{X},\ \phi(C_0, C_1) > \phi(C_0, C_2) \implies f(C_0, C_1) > f(C_0, C_2). \tag{2.8}$$

The target partition, $\mathcal{C}^\star$, which is model-based separated, corresponds to connected components in G.

We provide two examples to build intuition about model-based separation.

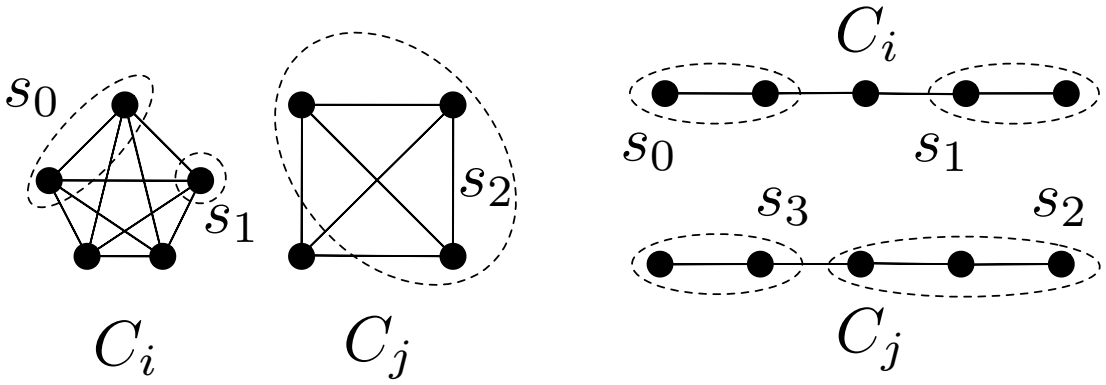
Example 4. Clique (Strict Separation). *Consider a graph $G = (\boldsymbol{X}, E)$ in which each connected component is a clique. Then if f separates G, every vertex in a connected component, C_i, is more similar to all other vertices in C_i than any vertex in connected component C_j, where similarity is defined by f. Thus, clique-structured connected components exactly capture strict separation (with a linkage of single/complete/average linkage) [26].*

Example 5. Chain. *Consider a graph $G = (\boldsymbol{X}, E)$ in which each connected component is chain-structured. According to Definition 1, two vertices that are part of the same chain but do not share an edge may be dissimilar under f even if f separates G. However, any two segments of the chain connected by an edge are similar under f.*

A visual illustration of both clique and chain style clusters is depicted in Figure 2.2. As we will see, chain structured connected components pose a challenge to existing incremental algorithms, something we resolve with GRINCH (Chapter 3).

A well studied separability assumption is δ-Separability [177]. Unlike the aforementioned definitions, δ-separability will be presented in terms of distances. We then will describe how to convert to similarities in the mapping to model-based separation. This assumption requires us to have vector data for which we can compute cluster centers.

Assumption 2. (δ-Separability [177]) *A dataset $\boldsymbol{X}$ satisfies δ-separation, with respect to some target clustering $\mathcal{C}^\star = \{C_1, C_2, \ldots, C_k\}$ if there exists centers $c_1^\star, \ldots c_k^\star$ such that for all $i \neq j$ $||c_i^\star - c_j^\star|| \geq \delta \cdot R$ where $R := \max_{l \in [k]} \max_{x \in C_l} ||x - c_l^\star||$.*



(a) Clique-shaped clusters.

(b) Chain-shaped clusters.

Figure 2.2: *Model-based separation*. Figure 2.2a shows two clique-shaped clusters with data points as vertices in a graph. If f separates the graph then $f(s_0, s_1) > \max[f(s_0, s_2), f(s_1, s_2)]$ because s_0 and s_1 form a connected subgraph. In Figure 2.2b, even if f separates the graph, it is possible for $f(s_0, s_1) < f(s_1, s_2)$. However, $f(s_1, s_2) < f(s_2, s_3)$.

Example 6. (Average Linkage & δ-separability) *Let $\boldsymbol{X}$ satisfies δ-separation with $\delta \geq 4$ for all metrics and $\delta \geq \gamma = 18$ for the ℓ_2^2 distance. The linkage function, $f(\cdot)$, of average linkage (Example 2) with a pairwise similarity, $\texttt{sim}(\cdot, \cdot)$, of negative distance satisfies model-based separation for dataset $\boldsymbol{X}$.*

Proof. See Section 2.8.

2.5.2 Noisy Model-based Separation

While model-based separation allows some additional flexibility compared to strict separation, it is overly rigid in its assumption that *every* point in a cluster has some point in their cluster that is closer than every point outside of their cluster. We propose a loosening of this restriction to allow *some* points to have nearest neighbors outside their clusters, which we refer to as *noisy model-based separation*.

Intuitively, noisy model-based separation says that a point x in a ground truth cluster $C^\star$ may have a nearest neighbor x' such that x' is not in $C^\star$, provided that there exists another

point $x'' \in C^\star$ whose nearest neighbor is x. Formally, we need to make an additional restriction on the linkages that separate this data:

Assumption 3. (Noisy Model-based Separation) *Let $G = (\boldsymbol{X}, E)$ be a graph with connected components $\mathcal{C}^\star$. Let $f : \mathcal{P}(\boldsymbol{X}) \times \mathcal{P}(\boldsymbol{X}) \to \mathbb{R}^+$ be a symmetric linkage function that computes the similarity of two groups of vertices. Let $\phi : \mathcal{P}(\boldsymbol{X}) \times \mathcal{P}(\boldsymbol{X}) \to \{0, 1\}$ be a function that returns 1 if the union of its arguments is a connected subgraph of G. The function f separates $\boldsymbol{X}$ if $\forall C_0,\ C_1,\ C_2 \subseteq \boldsymbol{X}$ either:*

- $\phi(C_0, C_1) > \phi(C_0, C_2) \Longrightarrow f(C_0, C_1) > f(C_0, C_2)$*; or*
- $|C_0| = |C_1| = |C_2| = 1$*,* $\phi(C_0, C_1) > \phi(C_0, C_2)$*,* $f(C_0, C_2) > f(C_0, C_1)$*,* $\exists x \in \boldsymbol{X}$ *s.t.* $\phi(C_0, \{x\}) = 1$ *and* $C_0 = \text{argmax}_{x' \in \boldsymbol{X}} f(\{x\}, \{x'\})$

In Figure 2.3, we give an example of two closely datasets one that satisfies model-based separation and the other that satisfies noisy model-based separation. The figure shows the linkage functions between pairs of points. In the noisy model-based separated data, we observe that the node b has d as its nearest neighbor despite being in different clusters, but that there exists another point c which has b as its nearest neighbor.

2.6 Algorithms

In this section, we review several clustering algorithms that are the basis of this work.

2.6.1 Hierarchical Agglomerative Clustering

Given a dataset $\boldsymbol{X}$, agglomerative methods work in a sequential round-based fashion, merging together clusters from the previous round. HAC is perhaps the best known agglomerative method. Each round of HAC (or, equivalently, *level* of the tree) is a flat clustering of $\boldsymbol{X}$; we denote the clustering from the i^{th} round as $\mathcal{C}^{(i)}$. The initial round of the algorithm begins with each point in a separate cluster $\mathcal{C}^{(0)} = \{\{x\} \mid x \in \boldsymbol{X}\}$. In each round, the two

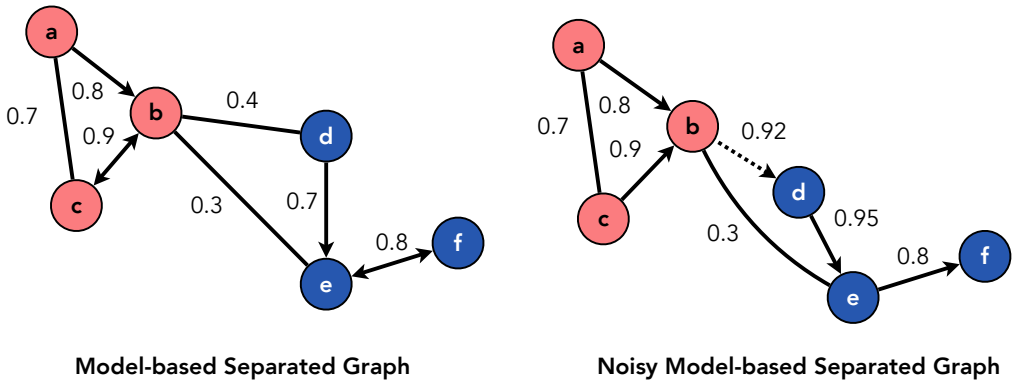


Figure 2.3: **Model-based Separation**. The node colors indicate the ground truth clusters. Edge directions indicate nearest neighbor relationships. Edge weights are symmetric similarities. (Left) An example of a graph that satisfies model-based separation. (Right) A slightly modified graph that only satisfies noisy-model-based separation. Observe how the nearest neighbor of b is the node d which is in another cluster.

clusters among those in the previous round that are most similar according to the linkage function are merged together. We use $\mathcal{C}^{(i)} = \{C_1, \cdots, C_K\}$ to refer to round i.

$$C, C' = \underset{B,B' \in \mathcal{C}^{(i-1)} \times \mathcal{C}^{(i-1)}}{\operatorname{argmax}} f(B, B') \tag{2.9}$$

$$\mathcal{C}^{(i)} = \left(\mathcal{C}^{(i-1)} \cup \{C \cup C'\}\right) \setminus \{C, C'\}. \tag{2.10}$$

This process continues until a complete tree or until a desired number of clusters is discovered, a threshold on similarity is met, or a full tree is built. Pseudocode for HAC is in Algorithm 1. Let L be the number of rounds of the algorithm, a hierarchical clustering is formed by $\cup_{i=1}^{L} \mathcal{C}^{(i)}$.

2.6.2 Reciprocal Nearest Neighbor

For certain linkage functions, we can design more efficient algorithms that will produce the same tree structure as Algorithm 1.

Algorithm 1 Hierarchical Agglomerative Clustering (HAC)

Input: $\boldsymbol{X}$: Dataset , f: Linkage
Output: $\mathcal{C}^{(0)}, \cdots, \mathcal{C}^{(i)}$: A sequence of flat clusterings that corresponds to a hierarchical clustering of $\boldsymbol{X}$
$\mathcal{C}^{(0)} \leftarrow \{\{x_1\}, \ldots, \{x_N\}\}$
$i \leftarrow 0$
while $|\mathcal{C}^{(i)}| > 1$ **do**
    $i \leftarrow i + 1$
    $C, C' = \text{argmax}_{B,B' \in \mathcal{C}^{(i-1)} \times \mathcal{C}^{(i-1)}} f(B, B')$
    $\mathcal{C}^{(i)} = \left(\mathcal{C}^{(i-1)} \cup \{C \cup C'\}\right) \setminus \{C, C'\}$.
return $\mathcal{C}^{(0)}, \cdots, \mathcal{C}^{(i)}$

Definition 7. (**Reducibility** [55]) *A linkage function f is said to be reducible if for any clusters in round i of HAC, $C, C', C'' \in \mathcal{C}^{(i)}$, s.t., $C \cup C'$ are reciprocal nearest neighbors, $f(C, C') > \max(f(C, C''), f(C', C''))$, and $C \cup C' \in \mathcal{C}^{(i+1)}$, then $f(C \cup C', C'') \leq \max(f(C, C''), f(C', C''))$.*

More efficient algorithms for linkage functions that satisfy reducibility are well established [55, 181, 216, 217].

Reciprocal nearest neighbors are defined as two clusters C, C' in a clustering $\mathcal{C}$ which are most similar to one another than any other cluster. The reciprocal nearest neighbor algorithm (RecipNN) [217] builds the clustering of round $\mathcal{C}^{(i)}$ by merging all pairs of reciprocal nearest neighbors in $\mathcal{C}^{(i)}$:

$$\mathcal{R}^{(i)} = \{(C, C') \mid C' = \underset{C'' \in \mathcal{C}^{(i-1)} \setminus \{C\}}{\text{argmax}} f(C, C'') \ \wedge C = \underset{C'' \in \mathcal{C}^{(i-1)} \setminus \{C'\}}{\text{argmax}} f(C', C'')\} \quad (2.11)$$

$$\mathcal{C}^{(i)} = \left(\mathcal{C}^{(i-1)} \cup \{C \cup C' | (C, C') \in \mathcal{R}^{(i)}\}\right) \setminus \{C'' | C'' \in \{C, C'\}, (C, C') \in \mathcal{R}^{(i)}\}. \quad (2.12)$$

For linkage functions that are reducible, this algorithm will produce the same hierarchical clustering as hierarchical agglomerative clustering [217].

Interestingly, both HAC and RecipNN will be able to correctly cluster model-based separated data. However, neither one can correctly cluster noisy model-based separated data.

Algorithm 2 Reciprocal Nearest Neighbor Algorithm (RecipNN)

Input: $\boldsymbol{X}$: Dataset , f: Linkage
Output: $\mathcal{C}^{(0)}, \cdots, \mathcal{C}^{(i)}$: A sequence of flat clusterings that corresponds to a hierarchical clustering of $\boldsymbol{X}$
$\mathcal{C}^{(0)} \leftarrow \{\{x_1\}, \ldots, \{x_N\}\}$
$i \leftarrow 0$
while $|\mathcal{C}^{(i)}| > 1$ **do**
  $i \leftarrow i + 1$
  $\mathcal{R}^{(i)} = \{(C, C') \mid C' = \text{argmax}_{C'' \in \mathcal{C}^{(i-1)} \setminus \{C\}} f(C, C'') \wedge C = \text{argmax}_{C'' \in \mathcal{C}^{(i-1)} \setminus \{C'\}} f(C', C'')\}$
  $\mathcal{C}^{(i)} = (\mathcal{C}^{(i-1)} \cup \{C \cup C' | (C, C') \in \mathcal{R}^{(i)}\}) \setminus \{C'' | C'' \in \{C, C'\}, (C, C') \in \mathcal{R}^{(i)}\}$
return $\mathcal{C}^{(0)}, \cdots, \mathcal{C}^{(i)}$

2.6.3 Affinity Clustering

Affinity clustering [31] is another round-based algorithm, which is inspired by the classic minimum spanning tree algorithm, Borůvka's algorithm[49]. As in HAC and RecipNN, Affinity builds a flat partition in each round round. In round i of Affinity, each cluster C from round $i-1$ finds its nearest neighbor. Clusters for the round i are formed by merging together all clusters that are connected to one another by the nearest neighbor relation. Let's now more rigorously define what we mean by connected. We call the set of connected clusters an *affinity component*:

Definition 8. (Affinity Component) *Clusters $C_j, C_k \in \mathcal{C}$ are defined to be part of the same* affinity component *according if there exists a path $P \subseteq \mathcal{C}$ defined as $\{C_j = C_{s_0}, C_{s_1}, C_{s_2}, \ldots C_{s_{R-1}}, C_{s_R} = C_k\}$, where: $C_{s_{r-1}} = \text{argmax}_{C \in \mathcal{C} \setminus \{C_{s_r}\}} f(C_{s_r}, C)$ and/or $C_{s_r} = \text{argmax}_{C \in \mathcal{C} \setminus \{C_{s_{r-1}}\}} f(C_{s_{r-1}}, C)$. We use $\mathsf{AC}(C_j, C_k, \mathcal{C}) = 1$ to denote that C_j and C_k are in the same affinity component.*

Inference at round i works by merging all clusters in the same affinity component together. Computationally, this is building a graph with nodes as the affinity components from the previous round and edges between pairs of nodes that are nearest neighbors. We define, $\mathsf{AC}(C_j, \mathcal{C})$, as a union of all sub-clusters in $\mathcal{C}$ that are within the affinity component of C_j, i.e.,

$$\mathsf{AC}(C_j, \mathcal{C}) := \bigcup_{\substack{C \in \mathcal{C}, \\ \mathsf{AC}(C_j, C, \mathcal{C})=1}} C. \tag{2.13}$$

Thus, $\mathsf{AC}(C_j, \mathcal{C}^{(i-1)})$ is a new cluster, created by taking a union of all clusters from round $i-1$ that are in the sub-cluster component of C_j. We create the flat partition at round i, $\mathcal{C}^{(i)}$, as the set of all of these newly found clusters:

$$\mathcal{C}^{(i)} := \{\mathsf{AC}(C, \mathcal{C}^{(i-1)}) \,|\, C \in \mathcal{C}^{(i-1)}\}. \tag{2.14}$$

Algorithm 3 Affinity Clustering

Input: $\boldsymbol{X}$: dataset , f: Linkage
Output: $(\mathcal{C}^{(0)}, \mathcal{C}^{(1)}, \ldots)$: One flat partition per round
$\mathcal{C}^{(0)} \leftarrow \{\{x\} \mid x \in \boldsymbol{X}\}$
$i \leftarrow 0$
while $|\mathcal{C}^{(i)}| > 1$ **do**
    $i \leftarrow i + 1$
    $\mathcal{C}^{(i)} := \{\mathsf{AC}(C, \mathcal{C}^{(i-1)}) \,|\, C \in \mathcal{C}^{(i-1)}\}$ where $\mathsf{AC}(C, \mathcal{C}) := \bigcup_{\substack{C' \in \mathcal{C}, \\ \mathsf{AC}(C, C', \mathcal{C})=1}} C'$
return $(\mathcal{C}^{(0)}, \ldots, \mathcal{C}^{(i)})$

2.7 Overview of Related Work

Clustering is a fundamental task in machine learning and a widely studied unsupervised learning problem. We attempt to provide a high level overview of key topics and approaches in clustering with high level overviews of data assumptions, clustering models, and algorithms for discovering flat, hierarchical, and DAG-structured clusterings in the offline (batch), incremental/online, and dynamic (addition/deletion) setting. We also refer readers to several excellent texts for more background on clustering [9, 153].

2.7.1 Flat Clustering

Flat clustering methods can be categorized as approaches optimizing a particular objective (e.g., K-Means or correlation clustering) or algorithmic approaches (e.g., DBSCAN)

as well as whether the clustering produced assigns each point to a single cluster (hard assignment) or not (soft assignment).

Frequently used objectives for flat clustering include K-Means, K-Center, and K-Median [21, 27, 28, 41, 52, 115, 128, 140, 143, 144, 153, 191, 195, 244, 264, 270, 270, 271, inter alia], correlation clustering [7, 10, 12, 14, 16, 30, 67, 78, 98, 162, inter alia], spectral clustering[25, 80, 100, 116, 186, 222, 250, 289, inter alia], affinity propagation [118, 249, 252, 278], mean-shift [76, 77, 86, 119, 254] and others [53, 157, 176, 230].

Algorithms for optimizing these objectives include approaches for initialization [8, 22, 24, 270], scaling to large numbers of clusters [90, 103, 109, 295], using coresets [8, 74, 142, 244], stochastic gradient based methods [50, 246] and others [28, 264]. Most closely related to this thesis, there is a multitude of work on optimizing these objectives in the online setting [12, 15, 39, 40, 66, 71, 79, 82, 185, 194, 215, 307, inter alia].

Density-based methods such as DBSCAN [112] are also widely studied. There are a number of online/incremental/streaming and hierarchical variants of these methods [20, 56, 57, 154]. There is also a close relationship between these methods and single linkage hierarchical clustering as well as hierarchical clustering with linkage functions more generally [72].

Closely related to the aforementioned clustering objectives are probabilistic models for clustering such as mixture models. Inference techniques for these models include scalable, distributed approaches of particular interest to this thesis [11, 105, 127, 149, 220, 285, 295, 296, inter alia]. It is important to note work on microclustering and its importance to entity resolution [35, 36, 37, 258]. These works introduce prior distributions such that the number of clusters grows sublinearly in the size of the dataset.

There is also a significant amount of work combining clustering and representation learning [47, 61, 62, 65, 158, 179, 247, 265, 287, 290, 297, inter alia]. These works learn representations of data (e.g., text or images) in such a way that the embedding space has meaningful cluster structure.

2.7.2 Hierarchical and DAG-Structured Clustering

Hierarchical clustering can be broadly categorized into algorithmic approaches and their approximations [1, 56, 97, 114, 121, 160, 174, 217, 218, 281, 284, 300, inter alia], approaches aimed at recovering data following a particular separation assumption [26, 29, 170, inter alia], and approaches optimizing a particular cost or objective [68, 69, 70, 83, 94, 106, 151, 214, 277, inter alia].

Most closely related to this thesis is work on building hierarchical clusterings in an online, incremental, or streaming manner [97, 170, 174, 203, 210, 211, 223, 224, 235, 284, 300]. BIRCH, one of the most widely used algorithms, builds a tree in a top down fashion splitting nodes under a condition on the mean/variance of the points assigned to a node [121, 300]. The BICO algorithm of Fichtenberger et al. [114] run BIRCH on coresets extracted from a large data stream. Widyantoro et al. [284] use a bottom-up approach in the incremental setting. Kranen et al [174] build an adaptive index structure for streaming data that allows for weighting points based on their recently of arrival. Menon et al [203] present an approach that also uses tree re-arrangements. Those re-arrangements remove the ancestors of the nearest neighbor of the newly added point and rebuild the remaining subtrees with HAC. Concurrent with this thesis work, Rajagopalan et al [235] present hyperplane-based methods that are invariant to the data ordering.

Our work is also closely related to the tree-based clustering methods proposed by [26] and [29]. These methods also build a tree structure in a bottom up manner using round-specific thresholds to determine the mergers. This work, in fact, uses less restrictive separation models than those considered in this thesis. Note that this work [29] uses a *particular* linkage function to discover tree structures that contain the target partition, given the more flexible separation model. Our model-based separation analysis, on the other hand, presumes a linkage function with the particular properties is given. We note that the more robust linkage functions from this related work do come with additional an computational expensive.

A variety of objective functions have been proposed for hierarchical clustering. Some work has use integer linear programming to perform hierarchical agglomerative clustering [126]. Notably, Dasgupta's cost function [94] has widely studied [68, 69, 70, 83, 214, 277]. The cost, which is defined as the sum over all pairs of datapoints of the similarity of the pair multiplied by the number of leaves of the least common ancestor of the pair, prefers trees which place similar points together near the leaves of the structure. Hierarchical agglomerative clustering has also been shown to be effective for k-means [136].

Modeling distributions over tree structures has been the subject of a large body of work, including various types of Bayesian non-parametric models [43, 124, 151, 169, 227, inter alia]. Bayesian non-parametric models typically define a posterior distribution over tree structures given data such as diffusion trees [169, 221], coalescent models [51, 151, 262], and in the case of grouped data, the nested Chinese restaurant processes [43] and nested hierarchical Dirichlet processes [227]. Other models, such as tree structured nested sticking breaking, provide a distribution over a different class of tree structures, one for which data can sit at internal nodes [124]. Factor graph-based distributions over tree structures such as [283] on the other hand support a flexible class of distributions over tree structures as in our approach. Inference in factor graph models as well as many of the Bayesian non-parameteric models is typically approximate or performed by sampling methods. Bayesian hierarchical clustering is a recursive, probabilistic, hierarchical model for data [146], which is related to agglomerative methods [181].

Scalability is widely considered in hierarchical clustering [31, 105, 106, 121, 152, 159, 160, 213, 219, 225, 243, 291, inter alia]. Dhulipala et al [101] propose efficient, near linear time algorithms for graph-based data. Abboud et al [1] propose LSH-based approaches for Ward and average linkage HAC. Yaroslavtsev and Vadapalli [291] use a graph sparsification approach along with parallel minimum spanning tree approach to achieve provably good approximate minimum spanning trees. Other work has use techniques to reduce the number of distance computations required to perform hierarchical clustering [111, 175, 248]. There

is also a multitude of work on distributed methods [31, 121, 159, 160, 213, 225, 243, 291, inter alia].

The discovery of DAG-structured clusterings has been considered by previous work as PoClustering [187, 188] (poset clustering), overlapping hierarchical clustering (OHC) [156], and others [33, 34, 59, 88, 102, 120, 155, 173, 201, inter alia]. In our empirical comparison, we compare to OHC which shares a similar structure to methods such as PoCluster [187, 188] and CLIXO [173], in their sequential consideration of ordered pairwise similarities. Pyramidal clustering [34, 102] represent a special case of DAG-structured clustering where nodes have at at most two parents. In the same way the relationship between ultrametrics and tree structures has been explored [13, 60, 84], theoretical work has considered the relationship between different kinds of metrics and DAG-structured clusterings as well as more general representational capacity considerations [34, 59, 88, 120, 201].

Mixed membership models such as sparse dictionary learning [197, inter alia] and latent feature models [135, inter alia] produce an assignment of points to overlapping clusters. These approaches typically attempt to reconstruct a data matrix and use the overlapping clusters to capture different components of each data point. Our work, on the other hand, represents alternative clusters, where each point is an equal member of its clusters. Our work differs from topic models and related models that build tree and DAG structures [227, 298, inter alia] in that we do not operate on grouped data.

Other methods that allows data to exist simultaneously in multiple clusters include ego-splitting based approaches [110]. Ego-splitting methods operate on graph-based data and create duplicate copies of certain nodes in the graph to allow the same node to exist in multiple clusters. We note that these approaches are limited to flat, not nested structures.

Geometric embeddings such as order [272] and box [274] embeddings can be used to represent partially ordered sets and DAGs. Directly representing every member of the powerset of a dataset using these methods is computationally infeasible and so would not be a reasonable alternative. However, these methods can also be used to define distributions

over the powerset. Crucially, these representations are closed under intersection and given a collection of elements, all subsets of the elements can be represented by combining the geometric embeddings of the base elements.

There is also work that tries to simultaneous learn representations of data and form a hierarchical clustering of the data [64, 130, 273, 305].

Finally, it is important to note related work in efficient data structures for operating on dynamic graphs [2, 3, 4, 5, 6, 17, 117, 125, 147, 147, 148, 150, 238, 240, 256, 260, 261, 266, inter alia]. This work allows for graph connectivity, minimum spanning trees, and other properties be maintained efficiently in the presence of node and edge additions and deletions.

2.8 Proofs and Additional Details

Example 6 (Average Linkage & δ-separability) *Let $\boldsymbol{X}$ satisfies δ-separation with $\delta \geq 4$ for all metrics and $\delta \geq \gamma = 18$ for the ℓ_2^2 distance. The linkage function, $f(\cdot)$, of average linkage (Example 2) with a pairwise similarity, $\texttt{sim}(\cdot, \cdot)$, of negative distance satisfies model-based separation for dataset $\boldsymbol{X}$.*

Proof. To see why, lets begin with the metric case. We use c to refer to the center of cluster C. Using the triangle inequality we have that:

$$||c_i^* - c_j^*|| \leq \frac{1}{|X||Y|} \sum_{x \in X} \sum_{y \in Y} ||c_i^* - x|| + ||x - y|| + ||y - c_j^*|| \tag{2.15}$$

Re-arranging terms we have:

$$||c_i^* - c_j^*|| \leq \frac{1}{|X|} \sum_{x \in X} ||c_i^* - x|| + \frac{1}{|X||Y|} \sum_{x \in X} \sum_{y \in Y} ||x - y|| + \frac{1}{|Y|} \sum_{y \in Y} ||y - c_j^*|| \tag{2.16}$$

With further re-arrangements,

$$||c_i^* - c_j^*|| - \frac{1}{|X|} \sum_{x \in X} ||c_i^* - x|| - \frac{1}{|Y|} \sum_{y \in Y} ||y - c_j^*|| \leq \frac{1}{|X||Y|} \sum_{x \in X} \sum_{y \in Y} ||x - y|| \tag{2.17}$$

Since $||c_i^* - c_j^*|| \geq \delta \cdot R$, where (see Assumption 2), $R := \max_{i \in [k]} \max_{x \in C_i^*} ||x - c_i^*||$:

$$(\delta - 2) \cdot R \leq ||c_i^* - c_j^*|| - \frac{1}{|X|} \sum_{x \in X} ||c_i^* - x|| - \frac{1}{|Y|} \sum_{y \in Y} ||y - c_j^*|| \leq \frac{1}{|X||Y|} \sum_{x \in X} \sum_{y \in Y} ||x - y|| \tag{2.18}$$

X and Y may be clusters from any ground truth cluster. Now, let's consider the average linkage for two subsets from the same ground truth cluster. For $X \subset C_i^*$ and $X' \subset C_i^*$:

$$\frac{1}{|X||X'|} \sum_{x \in X} \sum_{x \in X'} ||x - x'|| \leq \frac{1}{|X||X'|} \sum_{x \in X} \sum_{x' \in X'} ||c_i^* - x|| + ||c_i^* - x'|| \leq 2R. \tag{2.19}$$

We need $2R < (\delta - 2)R$ to differentiate two sets from the same cluster from sets from different clusters. We can do this if $\delta > 4$.

We can repeat the same analysis if ℓ_2^2 distance is used. Using the relaxed triangle inequality [136] for ℓ_2^2, we have:

$$\|c_i^* - c_j^*\|_2^2 \leq 3 \left(\frac{1}{|X||Y|} \sum_{x \in X} \sum_{y \in Y} \|c_i^* - x\|_2^2 + \|x - y\|_2^2 + \|y - c_j^*\|_2^2 \right) \tag{2.20}$$

$$\|c_i^* - c_j^*\|_2^2 \leq 3 \left(\frac{1}{|X|} \sum_{x \in X} \|c_i^* - x\|_2^2 + \frac{1}{|X||Y|} \sum_{x \in X} \sum_{y \in Y} \|x - y\|_2^2 + \frac{1}{|Y|} \sum_{y \in Y} \|y - c_j^*\|_2^2 \right) \tag{2.21}$$

Re-arranging the above:

$$\frac{1}{3} \|c_i^* - c_j^*\|_2^2 - \frac{1}{|X|} \sum_{x \in X} \|c_i^* - x\|_2^2 - \frac{1}{|Y|} \sum_{y \in Y} \|y - c_j^*\|_2^2 \leq \frac{1}{|X||Y|} \sum_{x \in X} \sum_{y \in Y} \|x - y\|_2^2 \tag{2.22}$$

Since $\|c_i^* - c_j^*\|_2^2 \geq \delta \cdot R$ and by the definition of R,

$$(\frac{1}{3}\delta - 2) \cdot R \leq \frac{1}{3} \|c_i^* - c_j^*\|_2^2 - \frac{1}{|X|} \sum_{x \in X} \|c_i^* - x\|_2^2 - \frac{1}{|Y|} \sum_{y \in Y} \|y - c_j^*\|_2^2 \leq \frac{1}{|X||Y|} \sum_{x \in X} \sum_{y \in Y} \|x - y\|_2^2 \tag{2.23}$$

However, for any two subclusters $X, X' \subset C_i^*$ then we know that:

$$\frac{1}{|X||X'|}\sum_{x\in X}\sum_{x\in X'}\|x-x'\|_2^2 \leq 2\left(\frac{1}{|X|}\sum_{x\in X}\|c_i^*-x\|_2^2+ \leq \frac{1}{|X'|}\sum_{x\in X'}\|c_i^*-x'\|_2^2\right) \leq 4R. \tag{2.24}$$

And so if $(\frac{1}{3}\delta - 2)R > 4R$ (i.e., $\delta > 18$) we can differentiate sets that are from the same ground truth cluster from those that are of different clusters.

Using the above analysis, if we have a linkage function f which is the negative average distance between points, then with model-based separation condition will hold with the settings of δ given above, since any two subsets of data from the same cluster will have higher similarity than any two subsets of data from different clusters. ■

CHAPTER 3

TREE RE-ARRANGEMENTS FOR INCREMENTAL HIERARCHICAL CLUSTERING

In this chapter, we present efficient algorithms for incremental hierarchical clustering. We consider a setting in which we observe a dataset one point at a time and are asked to produce a tree structure that incorporates the newly arrived data at each time step. The core of our algorithmic approaches is the use of data-driven tree re-arrangements to non-greedily handle the stream of data. These tree re-arrangements allow us to efficiently reconsider a small handful of past decisions. We begin by presenting algorithms that use local re-arrangements in the form of rotations. We then present algorithms that use global tree re-arrangements.

3.1 Warm-Up: Greedy Algorithms

Let's begin by considering a simple (though not particularly effective) greedy algorithm for incremental hierarchical clustering. We observe the dataset $\boldsymbol{X}$ one point at a time, $x_1, \ldots, x_N$. We are asked to produce at time step i, a hierarchical clustering $\mathcal{T}_i$ for points $x_1, \ldots, x_i$ after the point x_i is observed.

To produce the tree $\mathcal{T}_i$, the greedy algorithm begins with $\mathcal{T}_{i-1}$. For the incoming point, x_i, we find the the nearest neighbor point, which we refer to as v, among the first $x_1, \ldots, x_{i-1}$ points, according to the linkage function:

$$v = \underset{x \in x_1, \ldots, x_{i-1}}{\operatorname{argmax}} f(\{x\}, \{x_i\}). \tag{3.1}$$

Algorithm 4 GREEDY

Input: x_i: the point to add, $\mathcal{T}_{i-1}$: the hierarchical clustering built on the previous $i-1$ points, f: a linkage function
Output: $\mathcal{T}_i$: a hierarchical clustering of points 1 to i.
$v = \text{argmax}_{x \in x_1, \ldots, x_{i-1}} f(\{x\}, \{x_i\})$
Let $\texttt{anc}(v, \mathcal{T}_{i-1}) := \{C \mid C \in \mathcal{T}_{i-1},\ \{v\} \subset C\}$
$\mathcal{T}_i \leftarrow \mathcal{T}_{i-1} \setminus \texttt{anc}(v, \mathcal{T}_{i-1})$
$\mathcal{T}_i \leftarrow \mathcal{T}_i \cup \{C \cup \{x_i\} | C \in \texttt{anc}(v, \mathcal{T}_{i-1})\}$
$\mathcal{T}_i \leftarrow \mathcal{T}_i \cup \{v,\ x_i\}$
return $\mathcal{T}_i$

The greedy algorithm creates a new node, which will have x_i and v as its children. This new node, which corresponds to the cluster $\{x_i, v\}$, becomes v's direct parent and makes v's former parent (in $\mathcal{T}_{i-1}$) becomes v's grandparent in $\mathcal{T}_i$. All of v's ancestors in $\mathcal{T}_{i-1}$ will change to now include x_i in $\mathcal{T}_i$. In the hierarchical clustering set-based notation, the algorithm can be summarized as:

$$\mathcal{T}_i = (\mathcal{T}_{i-1} \setminus \texttt{anc}(v, \mathcal{T}_{i-1})) \cup \{C \cup \{x_i\} \mid C \in \texttt{anc}(v, \mathcal{T}_{i-1}\} \cup \{\{x_i, v\}\}, \tag{3.2}$$

where $\texttt{anc}(v, \mathcal{T}_{i-1})$ the ancestors of v in $\mathcal{T}_{i-1}$, i.e., $\{C \mid C \in \mathcal{T}_{i-1},\ \{v\} \subset C\}$. We also provide pseudo-code for the greedy algorithm in Algorithm 4.

It is easy to see how this greedy algorithm will make mistakes, even when the data follows the most restrictive assumptions (and therefore is easiest to cluster). Suppose the first two data points, x_1 and x_2, are of the same ground-truth cluster and the third data point, x_3 is of a different cluster. Even if x_1 and x_2 are more similar to each other, x_3 necessary will select one of these two as its nearest neighbors (say GREEDY adds x_3 as a sibling of x_1). The ground-truth clustering, in which x_1 and x_2 are in a cluster without x_3, is not a tree consistent partition of the resulting tree (and any subsequent trees on the remaining portion of the dataset). We will see in the following two sections how we can recover from such mistakes with non-greedy tree re-arrangements.

3.2 Local Re-arrangements with Tree Rotations

To recover from the kinds of greedy mistakes of the aforementioned algorithm, we can apply local tree rearrangements. Inspired by classic self-balancing binary search tree algorithms, we implore a rotation operation which swaps a node and its aunt in the tree structure. We initially designed rotation operations for points in Euclidean space [170] and then generalized the operation to any linkage function [210].

As in the greedy algorithm, the rotation-based algorithm will create a binary tree, $\mathcal{T}_i$, from a binary tree, $\mathcal{T}_{i-1}$, and the newly observed point x_i. The rotation-based algorithm will insert x_i as a sibling of its nearest neighbor. Then, x_i will be rotated up the tree according to condition that encourages the linkage function similarity between a node and its sibling to be higher than between a node and its aunt.

Let v be the nearest neighbor leaf of x_i as defined in Equation 3.1. We apply the insertion step, as in the greedy algorithm to acquire $\mathcal{T}_i$, to place x_i as a sibling of v. We will then apply rotations to $\mathcal{T}_i$. A rotation is applied if:

$$f(v, \texttt{sib}(v)) < f(v, \texttt{aunt}(v)), \tag{3.3}$$

where the functions $\texttt{sib}(\cdot)$ and $\texttt{aunt}(\cdot)$ return the sibling and aunt of their input, respectively. In words, if the node achieves a higher score under f with its aunt than with its sibling, then the aunt and sibling should be swapped.

Let's now define this rotation operation in terms of the set-based notation. Observe that the only node that changes in the rotation is parent of v. The parent containing v and $\texttt{sib}(v)$ is removed and a node containing v amd $\texttt{aunt}(v)$ is created. A rotation operation applied to node v in $\mathcal{T}_i$ is defined as:

$$(\mathcal{T}_i \setminus \{\texttt{parent}(v)\}) \cup (\{v\} \cup \texttt{aunt}(v)) \tag{3.4}$$

Algorithm 5 ROTATE

Input: x_i: the point to add, $\mathcal{T}_{i-1}$: the hierarchical clustering built on the previous $i-1$ points, f: a linkage function
Output: $\mathcal{T}_i$: a hierarchical clustering of points 1 to i.
▷ Find nearest neighbor among the leaf data points
Let $\texttt{lvs}(\mathcal{T}_{i-1}) := \{C \mid C \in \mathcal{T}_{i-1},\ |C| = 1\}$
$V \leftarrow \text{argmax}_{C \in \texttt{lvs}(\mathcal{T}_{i-1})} f(\{x_i\}, C)$
▷ Find proper sibling via rotations
while $f(V, \{x_i\}) < f(V, \texttt{sib}(V))$ and V is not the root of $\mathcal{T}_{i-1}$ **do**
    $V \leftarrow \texttt{parent}(V)$
▷ Update the ancestors of V to include the newly arrived point x_i
Let $\texttt{anc}(V, \mathcal{T}_{i-1}) := \{C \mid C \in \mathcal{T}_{i-1},\ V \subset C\}$
$\mathcal{T}_i \leftarrow \mathcal{T}_{i-1} \setminus \texttt{anc}(V, \mathcal{T}_{i-1})$
$\mathcal{T}_i \leftarrow \mathcal{T}_i \cup \{C \cup \{x_i\} | C \in \texttt{anc}(V, \mathcal{T}_{i-1})\}$
$\mathcal{T}_i \leftarrow \mathcal{T}_i \cup \{V \cup \{x_i\}\}$
return $\mathcal{T}_i$

where $\texttt{parent}(\cdot)$ gives the parent of a node in the tree structure and $\texttt{aunt}(\cdot)$ gives the aunt (parent's sibling).

In our proposed method, we apply *recursive* rotations, checking the rotation condition (and applying rotations as needed) at the ancestors of the newly added point. Note that in applying rotations in practice, the tree structure need not be continuously altered / updated for each rotation. Instead, we can instead simply can test v and each of v's ancestors to find the place in which x_i would stop rotating and simply insert the point as a sibling of that node. Algorithm 5 describes this rotation-based algorithm.

This simple operation helps us recover from the aforementioned failure of the greedy algorithm in the simple example. Assume our dataset follows clique model-based separation (Example 4) and suppose x_1 and x_2 are in the same cluster and x_3 in a different ground truth cluster then $f(\{x_1\}, \{x_2\}) > f(\{x_1\}, \{x_3\})$. Algorithm 5 would proceed in the following way: x_1 is observed, x_2 is added as a sibling of x_1, x_3 has (wlog) x_1 as its nearest neighbor, a rotation is applied keeping x_1 and x_2 as siblings with x_3 as their aunt. The tree contains the ground-truth clustering at this point.

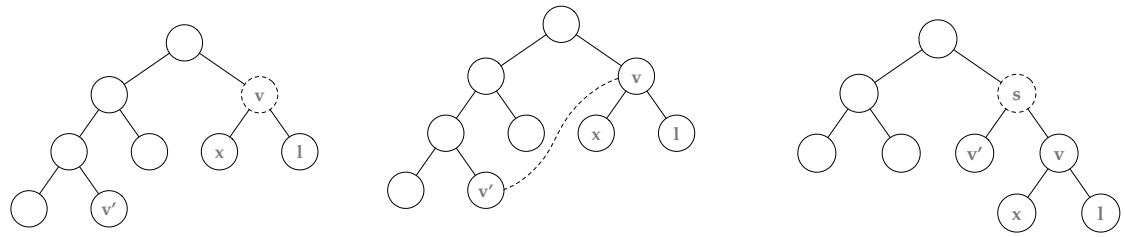
The rotation-based algorithm carries with it theoretical guarantees with respect to clique model-based separation separation (Example 4). This assumption states (1) the pairwise linkage function similarities between pairs of points from the same ground truth cluster are greater than similarity between any two points across clusters (2) the linkage function similarity between two sets of points from the same ground truth cluster is greater than between two sets that contain points from different ground truth clusters. Intuitively, (1) indicates that the nearest neighbor of each point will be a point from the same ground truth cluster. And when x_i arrives, if it is not the first point in its cluster, its nearest neighbor v will be from the same ground truth cluster. Condition (2) indicates that when rotations are applied they will either rotate the first point of a cluster out of subtree corresponding to a pure ground truth cluster or swap points that all belong to the same ground truth cluster. Condition (2) indicates that we will never perform rotation that swaps the sibling of a node and its aunt if the sibling and node are of one ground truth cluster and the aunt of different ground truth cluster. More formally, we make the following statement:

Theorem 1. *Let $\boldsymbol{X} = \{x_1, \ldots, x_N\}$ be a dataset and let f is a linkage function such that $\boldsymbol{X}$ is model-based separated by f with respect to a clique structured underlying graph (Assumption 1, Example 4). The hierarchical clustering $\mathcal{T}_i$ constructed via Algorithm 5 has dendrogram purity 1.0.*

Proof. See Section 3.6.

Theorem 1 indicates that after observing the entire dataset the ground truth partition will be a tree consistent partition:

Corollary 2. *Let $\boldsymbol{X} = \{x_1, \ldots, x_N\}$ be a dataset and let f is a linkage function such that $\boldsymbol{X}$ is model-based separated by f with respect to a clique structured underlying graph (Assumption 1, Example 4). The hierarchical clustering $\mathcal{T}_N$ constructed via Algorithm 5 contains the ground truth clustering $\mathcal{C}^\star$.*



(a) x is added to the cluster tree. (b) v finds its nearest neighbor v'. (c) v is grafted to v'.

Figure 3.1: *The* `graft` *subroutine*. Dotted lines denote new nodes and mergers. Before x is added, l and v' are in disjoint subtrees despite belonging to the same ground-truth cluster. The addition of x creates the subtree with root v and initiates the `graft`.

3.3 Global Re-Arrangements with Grafting and Restructuring

Even the ROTATE algorithm (Algorithm 5) cannot accurately recover any model-based separated (Assumption 1) data in the incremental setting. For instance, ROTATE cannot reliably recover chain structured clusters (Example 5).

3.3.1 Subtree Grafting

We introduce a non-local tree rearrangment called a *graft*, which facilitates the discovery of such chain-structured connected components. To insert the point x_i, we first we use the ROTATE algorithm to produce $\mathcal{T}_i$. Our proposed method will invoke a `graft` subroutine. At a high level, the `graft` procedure with respect to an ancestor node of x_i, $V \in \mathcal{T}_i$, searches $\mathcal{T}_i$ for a node V' that is both similar to V and dissimilar from its current sibling, $\texttt{sib}(V')$. If such a subtree is found, V' is disconnected from its parent and made a sibling of V. A visual illustration of a successful `graft` is depicted in Figure 3.1.

In detail, a `graft` searches the leaves of $\mathcal{T}_i$ that are not descendants of the node V for the nearest neighbor of V called L. Then it checks if the following holds:

$$f(V, L) > \max[f(V, \texttt{sib}(V)), f(L, \texttt{sib}(L))], \tag{3.5}$$

i.e., V and L prefer each other to their current siblings according to f. If the condition succeeds, we will re-arrange the tree structure detaching V and L from their old siblings and making V and L siblings under a common parent P such that P is the child of V's former parent. That is, a graft operation between V and L in tree $\mathcal{T}_i$ produces the following tree structure:

$$(\mathcal{T}_i \setminus (\texttt{ancs}(L, \mathcal{T}_i) \cup \texttt{ancs}(V, \mathcal{T}_i))) \cup \{V \cup L\}$$
$$\cup \{C \setminus L \mid C \in \texttt{ancs}(L, \mathcal{T}_i)\} \cup \{C \cup L \mid C \in \texttt{ancs}(V, \mathcal{T}_i)\}. \tag{3.6}$$

If the condition fails because L prefers its sibling to V, retest the condition at V and L's parent, $\texttt{parent}(L)$; if the condition fails because V prefers its sibling to L, then retest the condition at $\texttt{parent}(V)$ and L. Continue to check recursively until the condition succeeds or until the first time two nodes, V and L, are reached such that one is the ancestor of the other. The grafting subroutine is just part of our overall proposed algorithm. Pseudocode for the `graft` subroutine can be found in Algorithm 6.

Algorithm 6 `graft` $(V, \mathcal{T}_i, f)$

Input: V: a node, $\mathcal{T}_i$: the tree containing the first i datapoints, f: linkage function
Output: the ancestor of V where the grafting sub-routine ceased operation.
$L = \{\text{argmax}_{x \in X_i,\ x \notin V}\ f(x, V)\}$.
$V' \leftarrow \mathsf{lca}(V, L, \mathcal{T}_i)$ ▷ The least common ancestor of V and L
$S \leftarrow V$
while $V \neq V'$ **and** $L \neq V'$ **and** $\texttt{sib}(V) \neq L$ **do**
  if $f(V, L) > \max[f(V, \texttt{sib}(V)), f(L, \texttt{sib}(L))]$ **then**
    $Z \leftarrow \texttt{sib}(V)$
    $A \leftarrow \{C \cup L \mid C \in \texttt{ancs}(V, \mathcal{T}_i)\}$
    $B \leftarrow \{C \setminus L \mid C \in \texttt{ancs}(L, \mathcal{T}_i)\}$
    $\mathcal{T}_i \leftarrow (\mathcal{T}_i \setminus (\texttt{ancs}(L, \mathcal{T}_i) \cup \texttt{ancs}(V, \mathcal{T}_i))) \cup \{V \cup L\} \cup A \cup B$
    $\texttt{restruct}(Z, \mathsf{lca}(Z, V, \mathcal{T}_i), f)$
    break
  if $f(V, L) < f(L, \texttt{sib}(L))$ **then** $L \leftarrow \texttt{parent}(L)$
  if $f(V, L) < f(V, \texttt{sib}(V))$ **then** $V \leftarrow \texttt{parent}(V)$
if $V = S$ **then return** V' **else return** V

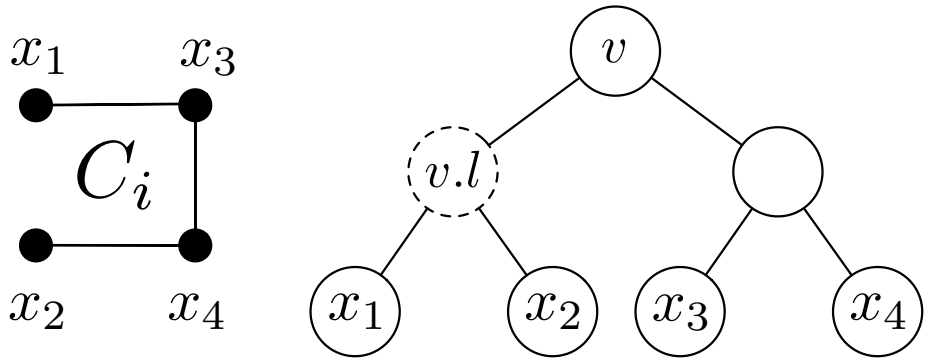


Figure 3.2: *Poorly structured tree*. Even though v's leaves form a connected subgraph in the graph on the left of the Figure (i.e., they all belong to cluster C_i), $v.l$'s descendant leaves, x_1 and x_2, are disconnected. An attempt to `graft` either x_1 or x_2 from a node whose descendants are not in C_i may succeed.

3.3.2 Tree Restructuring

While the `graft` subroutine facilitates discovery of chain-structured clusters, poorly structured trees are susceptible to having the `graft` subroutine disconnect previously discovered ground-truth clusters. As an example, consider Figure 3.2, in which $\texttt{lvs}(v)$ form the connected subgraph C_i (i.e., they all belong to the same ground-truth cluster). Consider v's left child, $v.l$, and its descendants, which form a *disconnected* subgraph. An attempt to `graft` either descendant, x_1 or x_2, may succeed, even when initiated from a node (not depicted) whose descendants are not connected to C_i. After such a `graft`, $\mathcal{T}$ cannot contain a tree-consistent partition that matches the ground-truth clustering.

Notice that a subtree can defend against spurious `grafts` by ensuring that each of its descendant subtrees is connected. For example, in Figure 3.2, if x_2 and x_3 were swapped, then each descendant subtree of v would be connected. Moreover, after such a swap, `grafts` from nodes whose descendants were not part of C_i would necessarily fail (assuming that f separates the graph).

During tree construction, the only step that can result in a connected subtree with disconnected descendants is the `graft` subroutine (see Section 3.4). We introduce the *restructure* (`restruct`) subroutine, which is performed after a successful `graft`, and

reorganizes a subtree with the intent of making each of its descendants connected. Let V be a node that was just grafted, Z be the previous sibling of V (i.e., before the graft) and let $R = \mathsf{lca}(Z, V)$ be the current least common ancestor of Z and V. `restruct` is initiated from V. First, the siblings of the ancestors of V (until R) are collected. Then, we find the node in the collection most similar to Z. If that node is more similar to Z than Z's current sibling (according to f), the two are swapped. The intuition here is that if a `graft` left Z and its new sibling disconnected, then the swap serves as a mechanism to restore the connectedness of Z's parent. Such swaps are attempted from the ancestors of Z until R. Pseudocode appears in Algorithm 7.

3.3.3 Grinch

Our proposed algorithm, GRINCH, which stands for: **G**rafting and **R**otation-based **INC**remental **H**ierarchical clustering combines the rotation, graft, and restructure tree rearrangement operations. The algorithm operates by first applying the ROTATE algorithm. Then we attempt a `graft` recursively from each ancestor of x_i. Each time a `graft` is successful, restructure the tree to group similar items together. Algorithm 8 presents the GRINCH algorithm.

Algorithm 7 `restruct`$(Z, R, \mathcal{T}_i, f)$

Input: Z: The former sibling of the node (V) which initiated the graft, R: The LCA of Z and V, $\mathcal{T}_i$: the hierarchical clustering built on the first i points, f: a linkage function
while $Z \neq R$ **do**
  $A \leftarrow \{\texttt{sib}(C) \mid a \in \texttt{ancs}(Z, \mathcal{T}_i) \backslash \texttt{ancs}(R, \mathcal{T}_i)\}$
  $M \leftarrow \text{argmax}_{C \in A} f(Z, C)$
  if $f(Z, \texttt{sib}(Z)) < f(Z, M)$ **then**
    $\triangleright$ Re-arrange tree structure such that $\texttt{sib}(Z)$ and M switch places.
    $\mathcal{T}_i \leftarrow \texttt{swap}(\texttt{sib}(Z), M)$
  $Z \leftarrow \texttt{parent}(Z)$

Algorithm 8 GRINCH $(x_i, \mathcal{T}_{i-1}, f)$

Input: x_i: the point to add, $\mathcal{T}_{i-1}$: the hierarchical clustering built on the previous $i-1$ points, f: a linkage function
Output: $\mathcal{T}_i$: a hierarchical clustering of points 1 to i.
$\mathcal{T}_i \leftarrow$ ROTATE$(x_i, \mathcal{T}_{i-1}, f)$
$V \leftarrow \texttt{parent}(x_i)$
while $V \neq \texttt{root}(\mathcal{T}_i)$ **do**
    $V \leftarrow \texttt{graft}(V, \mathcal{T}_i, f)$
return $\mathcal{T}_i$

3.4 Theoretical Analysis

We would like to understand what kinds of data GRINCH will be effective at clustering. We do so by making assumptions about the data and showing that GRINCH can recover this separated data. While data in practice may not satisfy these assumptions, this analysis shows that GRINCH will operate as expected when clusterings are clearly defined and provides insights into the representational power of GRINCH.

We will show that if data follows model-based separation (Assumption 1) GRINCH will recover a tree structure with the target partition as a tree consistent partition regardless of the data order.

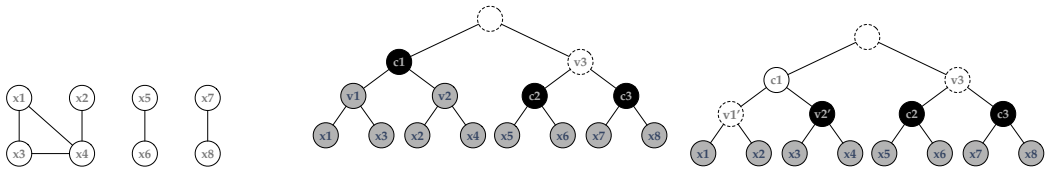
Theorem 3. *Given a dataset $\boldsymbol{X}$ and a linkage function f such that $\boldsymbol{X}$ is model-based separated with respect to f and latent graph $G = (\boldsymbol{X}, E)$, let $\mathcal{C}^\star$ be the target partition corresponding to the separated data.* GRINCH *builds a hierarchical clustering $\mathcal{T}$ such that $\mathcal{C}^\star$ is a tree consistent partition of $\mathcal{T}$ regardless of the ordering of points in $\boldsymbol{X}$.*

To prove this, we will show that the hierarchical clustering $\mathcal{T}_i$ produced by GRINCH after observing each point x_i, will satisfy two properties of *completeness* and *strong connectivity*. In particular, we will look at how each subroutine (rotation, graft, restructure) preserves these properties upon the arrival and processing of each point x_i. We first define these two properties:

Definition 9 (Connectivity). *Given a dataset $\boldsymbol{X}$ and a linkage function f such that $\boldsymbol{X}$ is model-based separated with respect to f and latent graph $G = (\boldsymbol{X}, E)$, a node $V \in \mathcal{T}$ V*

*is **connected** if V is a connected subgraph of G, **strongly connected** if every descendant of V in $\mathcal{T}$ is connected, and **maximal** strongly connected node if V is strongly connected and $\texttt{parent}(V)$ is not strongly connected. We say that the tree $\mathcal{T}$ satisfies **strong connectivity** if all connected nodes in $\mathcal{T}$ are strongly connected.*

Definition 10 (Completeness). *Given a dataset $\boldsymbol{X}$ and a linkage function f such that $\boldsymbol{X}$ is model-based separated with respect to f and latent graph $G = (\boldsymbol{X}, E)$, a node $V \in \mathcal{T}$ V is **complete** if V is a connected component in G. The tree $\mathcal{T}$ satisfies **completeness** if the set of connected components of G are a tree consistent partition of $\mathcal{T}$.*



(a) A graph $G = (\mathcal{X}, E)$.

(b) Strongly connected & complete.

(c) Complete only.

Figure 3.3: **Connectivity & Completeness**. A graph G with 3 connected components (Figure 3.3a). In Figure 3.3b and Figure 3.3c, black-bordered nodes are strongly connected, thick-black-border nodes are maximal, gray bordered nodes are connected (but not strongly) and nodes with dashed borders are disconnected. The tree in Figure 3.3b satisfies strong connectivity and completeness. The tree in Figure 3.3c does not satisfy strong connectivity because v_1 is disconnected.

First, we analyze the rotation operation. Rotations will preserve strong connectivity, but will not guarantee completeness.

Lemma 1 (Rotation Lemma). *Given a dataset $\boldsymbol{X}$ and a linkage function f such that $\boldsymbol{X}$ is model-based separated with respect to f and latent graph $G = (\boldsymbol{X}, E)$, let $\mathcal{T}_{i-1}$ be a hierarchical clustering produced by* GRINCH *using linkage function f over the first $i-1$ points, and $\mathcal{T}_i = \text{ROTATE}(x_i, \mathcal{T}_{i-1}, f)$ with $\texttt{lvs}(\mathcal{T}) = \boldsymbol{X}$, all nodes that were strongly connected in $\mathcal{T}_{i-1}$ are strongly connected in $\mathcal{T}_i$, i.e., rotations preserve strong connectivity.*

Now, we see how grafting preserves strong connectivity and completeness and how grafting from a node can help recover a target cluster.

Lemma 2 (**Grafting Lemma 1**). *Given a dataset $\boldsymbol{X}$ and a linkage function f such that $\boldsymbol{X}$ is model-based separated with respect to f and latent graph $G = (\boldsymbol{X}, E)$, let $\mathcal{T}_i$, which is built with linkage function f on the first i points of $\boldsymbol{X}$, satisfy strong connectivity and completeness, and let V be a node in $\mathcal{T}_i$ such that V is either a maximal strongly connected node or not strongly connected. The tree, $\mathcal{T}_i'$, resulting from the* `graft` *operation initiated from V, $\texttt{graft}(V, \mathcal{T}_i, f)$, satisfies strong connectivity and completeness.*

Lemma 3 (**Grafting Lemma 2**). *Given a dataset $\boldsymbol{X}$ and a linkage function f such that $\boldsymbol{X}$ is model-based separated with respect to f and latent graph $G = (\boldsymbol{X}, E)$, let $\mathcal{T}_i$, which is built with linkage function f on the points $x_1, \ldots, x_i$ corresponding to graph G_i, satisfy strong connectivity. Let V be strongly connected, but not complete with respect to the first i points, $V \subsetneq (C \cap \{x_1, \ldots, x_i\})$ for some C that is a connected component in G_i. Then a* `graft` *initiated from V returns a node V' such that V' is strongly connected and $V \subsetneq V' \subseteq C$.*

Lemma 4 (**Restructuring Lemma**). *Let $V \in \mathcal{T}_i$ be strongly connected. Let $A \in \texttt{ancs}(V)$ be the deepest connected ancestor of V such that: A is not strongly connected, and all siblings of the nodes on the path from V to A are strongly connected. Then* `restruct` *on inputs V and A restructures the tree rooted at A so that A satisfies strong connectivity.*

Using these lemmas we can prove Theorem 3. Grafting Lemma 2 shows that each connected component (in the latent graph, corresponding to a cluster in target partition) is represented by a node in the hierarchical clustering after each point is added. The other lemmas ensure that the strong connectivity and completeness properties are preserved so that Lemma 2 applies. See Section 3.6.6 for complete proof of Theorem 3 and proofs of each lemma.

3.4.1 Worst-case running time analysis

We measure the worst-case running time of the algorithm in terms of the number of linkage function calls. The GRINCH algorithm has four components: nearest neighbor search, rotations, grafting, and restructuring. Under particular assumptions (N-point metric space with expansion constant $c > 2$), exact nearest neighbor search is possible in $O(c^{12}\log(N))$ time per query where N is the number of data points [38]. Otherwise, exhaustive $O(N)$ time is required unless approximate nearest neighbor search methods are used. We will use $O(T)$ to refer to the time spent on nearest neighbor search. Let H be the height of the GRINCH tree, i.e., the length of the longest root to leaf path. There can be at most $O(H)$ rotations applied in a call to `insert`. In the extreme, `grafts` could be recursively applied moving all of the $N-1$ leaves (those other than the new point's sibling) one at a time in separate `grafts`. Since each graft involves a nearest neighbor search, there would be in the worst case $O(NT)$ computations from `grafts`. There can be at most $O(H)$ swaps made in a given `restruct` call, and each of these swaps involves considering $O(H)$ ancestors, resulting in $O(H^2)$ time overall. This results in $O(NT + H^2)$ time per data point. While there is no guarantee on the height of trees built with GRINCH, we find in practice that they are close to $\log^2 N$ as shown in Figure 3.4. If the trees had such a height, the running time would further be reduced to: $O(TN + \log^4(N))$ per point.

3.5 Experiments

We experiment with GRINCH to assess its scalability and accuracy. We begin by demonstrating that GRINCH outperforms other incremental clustering algorithms on a synthetic dataset. Observing that some of the steps of GRINCH are underutilized, we present four approximations of GRINCH's algorithmic components. We apply each approximation in turn and show that together they dramatically improve GRINCH's scalability without compromising its clustering quality. Then, we compare the approximate variant of GRINCH to state-of-the-art large scale hierarchical clustering methods. To showcase the flexibility

of GRINCH, we also provide experimental results in entity resolution, where the linkage function is learned. We provide analysis of the `graft` subroutine–GRINCH's distinguishing feature–and perform experiments to demonstrate the algorithm's robustness on adversarial data orderings. Finally, we show two use cases of GRINCH in cross-document coreference and in automatic knowledge-base construction.

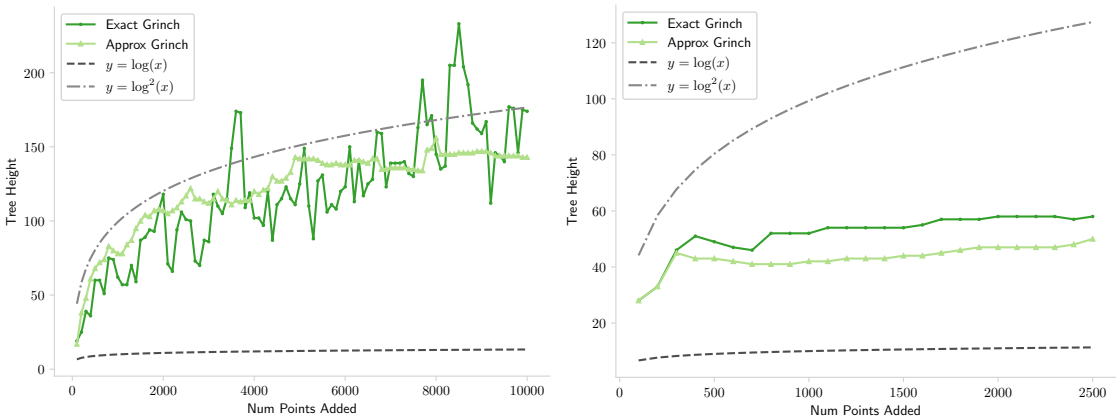
3.5.1 Synthetic Data Experiment

In our first experiment, we compare GRINCH to other incremental hierarchical clustering algorithms on a synthetic dataset in order to begin to understand GRINCH's empirical performance characteristics in a controlled manner. The data is generated so that it satisfies model-based separation with respect to cosine similarity. In particular, the dataset contains 2,500 10,000-dimensional binary vectors that belong to 100 clusters, with 25 points per cluster. Points in cluster k have bits $100(k-1)$ to $100k-1$ set randomly to 1 with probability 0.1. All other bits are set to 0. This way, across cluster points have cosine similarity 0 and within cluster points can have either 0 or non-zero cosine similarity. In other words, two points, x_1 and x_2, in the same cluster can appear to be dissimilar and end up in distant regions of the tree. The representation of each internal node in the GRINCH tree is the sum of the vectors of its descendant leaves. We compute the cosine similarity between two nodes v and v' as the cosine similarity between their aggregated vectors (we refer to this as *cosine linkage* in the following sections). We compare GRINCH, ROTATE and GREEDY.

The experimental results reveal that GRINCH achieves perfect dendrogram purity (1.0), which is expected given GRINCH's correctness guarantee. ROTATE achieves a dendrogram purity of 0.872 while GREEDY achieves 0.854. ROTATE and GREEDY do not construct trees of perfect purity because of their inability to globally rearrange a cluster hierarchy.

3.5.2 Scalability and Approximations

Some of the algorithmic steps of GRINCH, which are required to prove its correctness, are seldom invoked in practice. For example, and perhaps expectedly, a `graft` is unlikely



(a) ALOI (b) Synthetic

Figure 3.4: The height of trees built with exact and approximate (capping, single elimination, and single NN search) GRINCH are shown for a subset of ALOI and the Synthetic data described in Section 3.5.1. We notice that trees are fairly balanced and not chain structured. The height is close to (if not less than) to $\log^2 N$.

to succeed between two nodes close to the root of the tree. Therefore, we introduce a handful of approximations designed to have little effect on the quality of the clusterings constructed by GRINCH, but also designed to make the algorithm significantly faster in practice.

1. **Capping.** The recursive subroutines ROTATE, `graft`, and restruct, improve performance, but are also computationally expensive to check, and often are not applied. Moreover, we notice that tree rearrangements that occur close to the root do not have a significant, instantaneous effect on dendrogram purity. Therefore, we introduce *rotation*, *graft* and *restructure caps*, which prohibit rotations, grafts and restructures from occurring above a height, h.
2. **Single Elimination Mode.** The `graft` subroutine generally improves GRINCH's clustering performance, and is essential in attaining perfect purity on the synthetic dataset, but we find that `graft` attempts are rejected many more times than they are accepted. However, at times, we observe that a sequence of recursive `grafts` are accepted when initiated close to the leaves. Therefore, to limit the number of attempted `grafts` while retaining these `graft` sequences, we introduce *single elimination mode*. In this mode,

Approx.	DP	Time (s)	ALOI # Rotate	# Graft	# Restr.
GRINCH (No Approx).	0.533	85.371	7107	2435	1088
w/ Cap (100)	0.533	48.452	6495	2157	686
w/ Single Elimn	0.534	39.019	6574	1586	533
w/ Single NN	0.540	22.226	6441	1516	570
w/ no Restruct	0.538	14.292	6477	1634	0
w/ no Graft	0.506	12.748	6747	0	0
w/ no Rotate	0.442	14.793	0	0	0

Table 3.1: *Ablation.* Each row in the table represents GRINCH with the corresponding approximation applied in addition to all approximations contained in previous rows. The first 4 approximations significantly decreases the computational cost of GRINCH, but do not compromise DP. The ablation is performed for the first 5000 points of ALOI and the Synthetic datasets.

the recursive grafting procedure terminates after a `graft` between v and v' fails because both prefer their current siblings to a merge.

3. **Single Nearest Neighbor Searching.** GRINCH makes heavy use of nearest neighbor search under the linkage function f. Rather than perform nearest neighbor search anew for each `graft`, when a data point arrives, we perform a single k-NN search ($k \in [25, 50]$) and only consider these nodes during subsequent `grafts` (until the next data point arrives).
4. **Navigable Small World Graphs.** Instead of performing nearest neighbor computations exactly, we can perform them approximately. We employ a *navigable small world* nearest neighbor graph (NSW)–a data structure inspired by search in small world networks [166, 168, 282]. To find the nearest neighbor of a data point, x_i, in an NSW, begin at a random node, v. If the similarity between x_i and v is maximal among all neighbors of v, terminate; otherwise, move to the neighbor of v most similar to x_i. To insert a new data point, x_j, find its k nearest neighbors and add edges between those neighbors and a new data point [198]. Thus, NSWs are constructed online. We simultaneous construct a hierarchical clustering and an NSW over the data points stored in the tree's leaves.

To measure the effects of our approximations on the speed and quality of the resulting algorithm, we conduct the following ablation. We run GRINCH on our synthetically generated dataset as well as a random 5k subset of the ALOI [123] dataset and measure dendrogram purity, time, and the number of calls made to ROTATE, `graft` and `restruct`. We repeat the procedure multiple times, each time adding one of the following approximations, in order: capping, single elimination, single nearest neighbor search and approximate nearest neighbor search. Capping is performed at height 100. We also experiment with removal of the `graft` and ROTATE subroutines.

The result of the ablation is contained in Table 3.1. We observe that, for both datasets, each of the approximations reduces the computational cost of algorithm without effecting the resulting DP. However, once `grafts` are removed, the DP drops by 3% on ALOI and 12% on the synthetic datasets. When ROTATE is also removed, DP drops by an additional 6% and 2%, respectively.

Having verified that on a subset of ALOI our approximations improve scalability at little expense in terms of dendrogram purity, in the following experiments we report results for GRINCH in single elimination mode and with the rotation cap set to $h = 100$. We did not tune the cap parameter and used that same setting for all datasets. We mildly tune the number of nearest neighbors connected in NSW construction setting based on the dimensionality of the data, using values of 5 for CovType, 50 for ALOI, and 100 for the other datasets. This can be tuned based on the nearest neighbor performance of the structure (independent of the clustering algorithm).

3.5.3 Clustering Benchmarks

We compare GRINCH with the following 4 algorithms: **GREEDY** - an online hierarchical clustering algorithm that consumes one data point at a time and places it as a sibling of its nearest neighbor; **ROTATE** - an incremental algorithm that places a data point next to its nearest neighbor and then performs rotations until Equation 3.3 holds; **MB-HAC** - the

Approx.	DP	Time (s)	Synthetic # Rotate	# Graft	# Restr.
GRINCH (No Approx).	1.0	160.307	2558	578	203
w/ Cap (100)	0.993	164.328	2558	578	194
w/ Single Elimn	0.997	157.622	2523	526	184
w/ Single NN	0.993	83.014	2517	415	148
w/ no Restruct	0.993	82.262	2476	426	0
w/ no Graft	0.872	82.055	2259	0	0
w/ no Rotate	0.854	80.526	0	0	0

Table 3.2: *Ablation.* The same abalation as Table 3.1 on the Synthetic dataset.

mini-batch version of HAC, which keeps a buffer of size b, runs a single step of HAC using the data points in the buffer and then adds the next record to the buffer; **HAC** - best-first, bottom-up hierarchical agglomerative clustering and **PERCH** - a state-of-the-art large scale hierarchical clustering method.

We run each algorithm on 5 large scale clustering datasets: CovType, a datset of forest covertype, ALOI [123], a 50K subset of the Imagenet ILSVRC12 dataset [242] and the Speaker dataset [132], and a 100K subset of ImageNet containing all 17K classes not just the subset in ILSVRC12. Datasets have 500K, 50K, 100K, 36K, and 100K instances, respectively. We run each algorithm under two different linkage functions: average linkage and centroid cosine linkage. To compute the centroid cosine similarity between two nodes, v and v', first, for each node, compute the sum of the vectors contained at their descendant leaves. Then, compute the cosine similarity between the aggregated vectors.

Results are displayed in Table 3.3, where we record the dendrogram purity averaged over 5 replicates of each algorithm, where for each replicate we randomize the arrival order of the data. The table reveals that GRINCH–under both linkage functions–outperforms the corresponding versions of ROTATE and GREEDY on all datasets except for on the CovType dataset where the methods all seem to perform equally well. This underscores the power of the `graft` subroutine. GRINCH with approximate nearest neighbor search even outperforms PERCH, which uses exact nearest neighbor search, on ALOI. Recall that, unlike

the HAC variants, PERCH employs a specific linkage function. Seeing as the HAC variants outperform PERCH on Speaker suggests that the ability to equip various linkage functions can be advantageous. HAC is best on Speaker, but cannot scale to ALOI.

Algorithm	Linkage	CovType	ILSVRC12 (50k)	ALOI
GRINCH	Avg	0.43 ± 0.00	$\mathbf{0.557 \pm 0.003}$	0.504 ± 0.002
GRINCH	CS	0.43 ± 0.00	$\mathbf{0.544 \pm 0.005}$	0.499 ± 0.003
ROTATE	Avg	0.43 ± 0.01	0.545 ± 0.004	0.476 ± 0.004
ROTATE	CS	0.44 ± 0.01	0.513 ± 0.007	0.472 ± 0.003
GREEDY	—	0.44 ± 0.01	0.527 ± 0.004	0.435 ± 0.004
PERCH [170]	—	0.45 ± 0.004	0.53 ± 0.003	0.44 ± 0.004
PERCH-BC [170]	—	0.45 ± 0.004	0.36 ± 0.005	0.37 ± 0.008
MB-HAC [170]	Best	0.44 ± 0.005	0.43 ± 0.005	0.30 ± 0.002
BIRCH (incremental) [300]	-	0.44 ± 0.002	0.09 ± 0.006	0.21 ± 0.004
HAC [170]	Avg.	–	0.54	–
BIRCH (rebuild) [300]	-	0.44 ± 0.002	0.26 ± 0.003	0.32 ± 0.002
HDBSCAN [56]	-	**0.473**	0.414	**0.599**

Algorithm	Linkage	Speaker	ImageNet (100k)
GRINCH	Avg	0.480 ± 0.003	$\mathbf{0.065 \pm 0.000}$
GRINCH	CS	0.478 ± 0.003	0.062 ± 0.000
ROTATE	Avg	0.407 ± 0.003	0.063 ± 0.001
ROTATE	CS	0.406 ± 0.003	0.062 ± 0.000
GREEDY	–	0.317 ± 0.002	0.0589
PERCH [170]	–	0.37 ± 0.002	$\mathbf{0.065 \pm 0.000}$
PERCH-BC [170]	–	0.09 ± 0.001	0.03 ± 0.00
MB-HAC [170]	Best	0.01 ± 0.002	–
BIRCH (incremental) [300]	-	0.02 ± 0.002	0.02 ± 0.00
HAC [170]	Avg.	**0.55**	–
BIRCH (rebuild) [300]	-	0.22 ± 0.006	0.03 ± 0.00
HDBSCAN[56]	-	0.396	-

Table 3.3: Dendrogram Purity results for GRINCH and baseline methods. We compare two linkage functions: approximate average linkage (Avg) and cosine similarity linkage (CS).

3.5.4 Author Coreference

Bibliographic databases, like PubMed, DBLP, and Google Scholar, contain citation records that must be attributed to the corresponding authors. For some records, the attribution process is easy, but for many others, the identities of a publication's authors are ambiguous. For example, DBLP contains hundreds of citations written by different authors named "Wei Wang" that currently cannot be disambiguated [1]. Intuitively, author coreference datasets often exhibit chain like structures because a single citation written by a prolific author (perhaps in a short-lived collaboration) may only be similar to a small number of that author's other citations and dissimilar from the rest.

Following previous work, we train a linkage function to predict the likelihood that a group of citation records were all written by the same author [89, 255, 283]. We train our model, that utilizes features such as coauthor names, title, venue, year, etc., by running HAC and, at each step, use the model to predict the precision of merging two groups of records as in [180].

We compare the 5 HAC variants in author coreference on two datasets with labeled author identities: **Rexa** [89] and **PSU-DBLP** [141]. Evaluation is done using the pairwise F1-score of a predicted flat clustering against the ground-truth clustering. To compute pairwise F1-score, each pair of citations that appears in both the same ground-truth and predicted clusters is considered a true positive; each pair of citations that belongs to different ground-truth clusters but the same predicted cluster is considered a false positive. None of the authors represented in the test set, have any publications in the training set.

Figure 3.4 shows the precision, recall, and pairwise F1-score achieved by each method. The results show that GRINCH outperforms the other scalable methods on both datasets and even outperforms HAC on DBLP. This behavior may stem from overfitting of the learned linkage function, which is exploited by HAC; since GRINCH only approximates HAC,

[1] `https://dblp.uni-trier.de/pers/hd/w/Wang:Wei`

	Rexa			DBLP		
Algorithm	**Pre**	**Rec**	**F**	**Pre**	**Rec**	**F**
GRINCH	0.808	0.883	**0.844**	0.809	0.620	**0.701**
ROTATE	0.864	0.641	0.734	0.876	0.554	0.678
GREEDY	0.850	0.209	0.331	0.827	0.151	0.255
MB-HAC-Med.	0.807	0.881	**0.843**	0.375	0.631	0.461
MB-HAC-Sm.	0.922	0.333	0.483	0.697	0.151	0.247
HAC	0.805	0.887	**0.844**	0.741	0.600	0.664

Table 3.4: Precision, recall and F-Score of various methods on the Rexa and DBLP datasets.

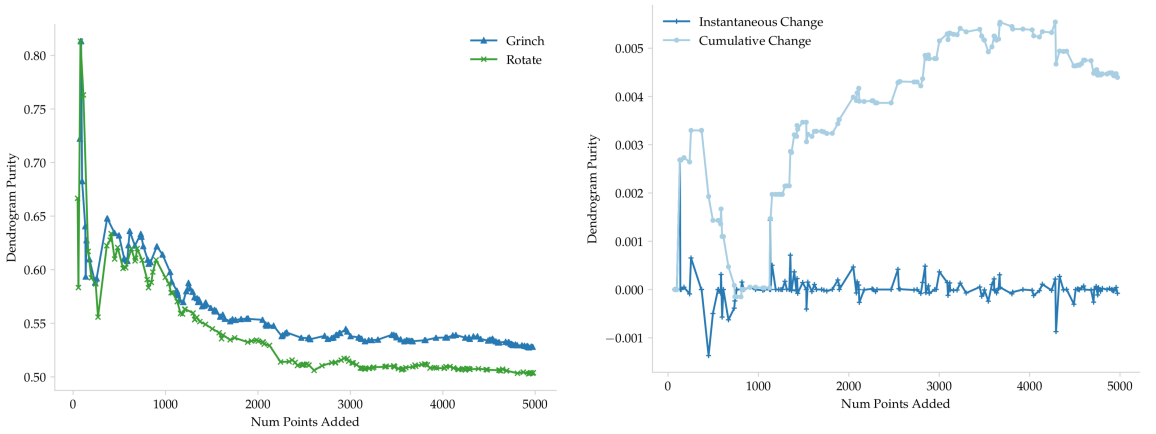
it can be thought of as a form of regularization. Again, GRINCH outperforms GREEDY and ROTATE on both datasets underscoring the importance of the ROTATE and `graft` procedures.

3.5.5 Significance of Grafting

The results above indicate that GRINCH, even when employing a number of approximations, constructs trees with higher dendrogram purity than other scalable methods in a comparable amount of time. Interestingly, GRINCH only differs from the algorithm ROTATE in its use of the `graft` (and subsequent `restruct`) subroutine. To better understand the significance of `grafting`, we compare GRINCH and the algorithm ROTATE on the first 5000 points of ALOI.

Figure 3.5a shows that dendrogram purity as a function of the number of data points inserted for both GRINCH and ROTATE and the first 5000 points of ALOI. Echoing the results above, by ~ 1000 points, GRINCH dominates ROTATE.

Figure 3.5b shows the instantaneous and cumulative change in dendrogram purity due to `grafts` made by GRINCH. That is, for the i^{th} data point, x_i, we record the dendrogram purity after x_i is inserted and rotations are performed (i.e., what would be executed by ROTATE). Then, we perform `grafting` (if appropriate) and record the dendrogram purity *after* all recursive `grafts` have been completed. The difference between the dendrogram purity after `grafting` and before `grafting` (but after rotations) is the instantaneous change



(a) Dendrogram purity per point.

(b) Instantaneous/cumulative change in DP due to `grafts`.

Figure 3.5: Figure 3.5a shows the dendrogram purity of two trees, one built by GRINCH and the other built by ROTATE, on the first 5000 points of ALOI. The dendrogram purity of the tree built GRINCH is greater than that of the tree built by ROTATE. Figure 3.5b plots the instantaneous and cumulative change in dendrogram purity due to `grafts`. While GRINCH achieves 3% larger dendrogram purity than ROTATE

in dendrogram purity due to `grafts`; the sum of instantaneous changes is the cumulative change.

Note the y-axis of Figure 3.5b, which reveals that even the most instantaneously significant `grafts` only lead to minute changes in dendrogram purity (of about 0.001). Moreover, after 5000 points, the cumulative change in dendrogram purity due to `grafts` is less than 0.005–hardly accounting for the difference in dendrogram purity between the tree built by GRINCH and the tree built by ROTATE (of 0.03). We conclude from these measurements that the increase in performance due to the `graft` subroutine is related to the rearrangement of small numbers of points. These rearrangements do not immediately have significant impact on dendrogram purity, but they do have significant long-term affects. To make this hypothesis more concrete, consider the case in which two dissimilar data points from the same cluster are split between two distant regions of the tree early on in clustering. The points are never merged (via a `graft`) and so each point draws a significant portion of the cluster's other data points to its location in the tree. This has dire consequences with respect

to dendrogram purity. If a `graft` is performed early on to correct the split, an adverse scenario like this can be averted.

3.5.6 Robustness

For completeness, we perform an experiment used in previous work to test an incremental clustering algorithm's robustness to data point arrival order [170]. In the experiment, a dataset is ordered in two specific ways. **Round-Robin** – Randomly determine an ordering of ground-truth clusters. Then, construct a data point arrival order such that the ith data point is a member of cluster $i \bmod K$, where K is the number of clusters and `mod` returns the remainder when its first argument is divided by its second. **Sorted** – Randomly determine an ordering of ground-truth clusters. All points of cluster C_i arrive before any point of cluster C_{i+1} arrives. As in previous work, we perform a robustness experiments with the ALOI dataset.

Table 3.5 shows that GRINCH achieves higher dendrogram purity than both PERCH and mini-batch HAC (with 2 different batch sizes) on data ordered using the Round Robin ordering scheme. Under this arrival order, MB-HAC performs poorly showing its lack of robustness. When the data is in Sorted order–which makes for easier clustering for MB-HAC–GRINCH outperforms PERCH and is competitive with MB-HAC.

Method	Round.	Sort.
GRINCH	0.503	0.457
PERCH	0.446	0.351
MB-HAC (5K)	0.299	0.464
MB-HAC (2K)	0.171	0.451

Table 3.5: Dendrogram Purities for adversarial arrival orders (ALOI).

3.5.7 Cross-Document Coreference

To further illustrate the use of GRINCH in entity resolution, we perform experiments in cross document coreference entity and event coreference. In this task, ambiguous mentions

of entities and events that appear throughout a corpus of documents are to be clustered into groups such that each group refers to the same real world entity or event. Cattan et al [63] present a state-of-the-approach approach, which (1) learns a pairwise similarity function between ambiguous mentions and (2) uses average-linkage agglomerative clustering with a similarity threshold to produce the predicted clustering. Of course, agglomerative clustering does not supports incremental addition of data. And so in our experiments, we attempt to answer the question of whether GRINCH can be used to achieve comparable performance to HAC. We select flat clusters from the tree built using GRINCH by cutting the tree at a given threshold.

The pairwise similarity function is a MLP which takes as input the concatenation of RoBERTa [190], embeddings of two mentions and their elementwise product. This induces an asymmetric similarity in fact.

We consider the 'end-to-end' setting of this task, in which we need to select the tokens in each document that correspond to a mention of the entities as well as performing the clustering of those entity mentions. Formally, given a corpus of documents $\mathcal{D}$, each document consists word tokens $w_1, \ldots, w_T$. Performing cross-document coreference requires predicting a set of mentions $\mathcal{M}$ where each $m \in \mathcal{M}$ is a sequence of tokens $w_i, w_{i+1}, \ldots, w_{i+k}$. There is a ground truth set of mentions $\mathcal{M}^\star$. We evaluate the performance of the predicted $\mathcal{M}$ compared to $\mathcal{M}^\star$ according to the MUC, B^3, CEAF metrics and their average (also known as CoNLL) [232]. For the ECB+ data, the documents are organized into a set of topics. The dataset assumes that each entity appears in only one topic. We evaluate in a setting where the topic of each document are assumed to be known.

Table 3.6 reports the results for this task. We find that GRINCH achieves results that are with 1 point of the F1 of HAC and yet is fully incremental.

	MUC	B3	CEAF-E	CoNLL
HAC	50.25	33.47	32.16	38.62
GRINCH	49.65	32.69	31.75	38.03

Table 3.6: **Cross Document Coreference on ECB+**: We report F1 for each metric.

3.5.8 Downstream Use Case: Automatic Knowledge-base Completion

We highlight a use of GRINCH in the larger system for automatic knowledge-base completion, which we presented in [92]. We use GRINCH to discover entity types (e.g., categories of entities such as *sports-people*, *religious entities*, etc).

Das et al [92] presents a case-based reasoning (CBR) system, which solves a new problem by retrieving 'cases' that are similar to the given problem, for automatic knowledge-base completion. Our approach predicts attributes for an entity by gathering reasoning paths from similar entities in the KB. Our probabilistic model estimates the likelihood that a path is effective at answering a query about the given entity. The parameters of our model can be efficiently computed using simple path statistics and require no iterative optimization. The reasoning path in the knowledge-base are represented by the entities and relations present on the path. However, parameterizing at the entity level leads to a large amount of sparsity in data. Instead, we form clusters of entities, representing entity types to deal with the data sparsity. Our simple model is non-parametric, growing dynamically as new entities and relations are added to the KB. And as such, we demonstrate the effectiveness of our model in an "open-world" setting where new entities arrive in an online fashion. In this setting, we rely on GRINCH to incrementally update the discovered entity types. Refer to Figure 3.6 for an illustration of the task and highlight description of the model.

We consider two particular experiments from [92]. The first we directly measure the impact of clustering on the model's performance. These results are summarized in Table 3.7. Across all metrics, we find that clustering dramatically increases performance of the model. Second, we show the performance of the model in the open world setting. We observe that

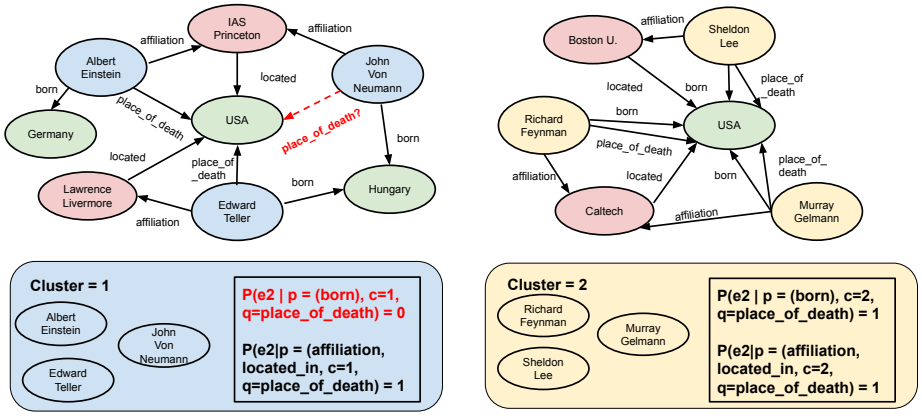


Figure 3.6: Given the query, (JON VON NEUMANN, PLACE_OF_DEATH, ?), our model gathers reasoning paths from similar entities such as other scientists. However, not all gathered paths work for a query e.g. the path ('BORN(x, y)') would not work for VON NEUMANN. This highlights the importance of learning path weights for *clusters of similar entities*. Even though 'BORN_IN' could be a reasonable path for predicting PLACE_OF_DEATH, this does not apply for VON NEUMANN and other scientists in his cluster. The precision parameter of the path given the cluster helps in penalizing the 'BORN_IN' path. Note that the node USA is repeated twice in the figure to reduce clutter. Figure from [92].

when using GRINCH for clustering we are able to match the performance of re-building the model at each time step using hierarchical agglomerative clustering in Figure 3.7. We show example entity types discovered in Figure 3.8.

	Das et al [92] w/ clustering	**Das et al [92] w/o clustering**
HITS@1	**0.42**	0.29
HITS@3	**0.46**	0.36
HITS@10	**0.51**	0.45
MRR	**0.45**	0.34

Table 3.7: **Impact of clustering on WN18RR**. We observe that across all metrics clustering entities into entity types provides a dramatic increase in performance.

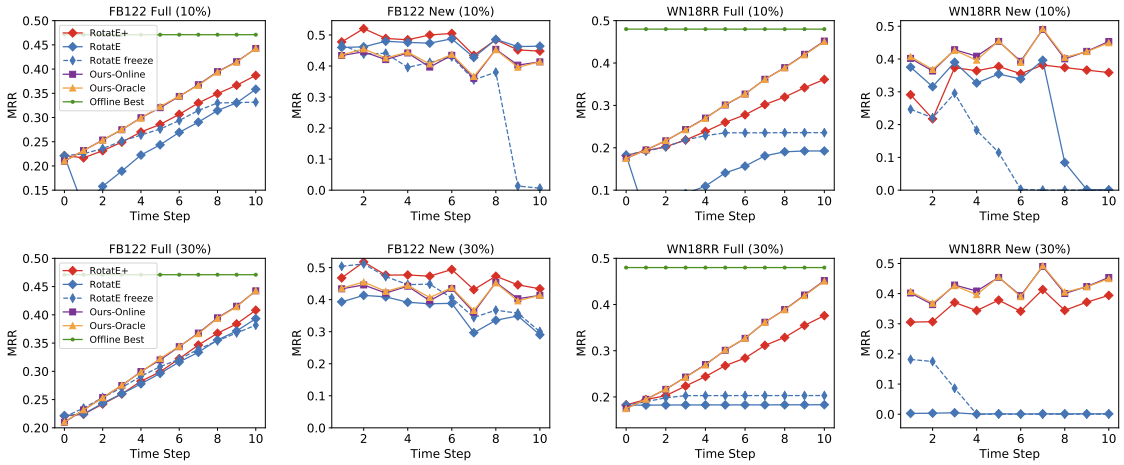


Figure 3.7: Results for open-world setting in CBR when trained with 10% (top row) and 30% (bottom row) of already seen edges. Our online method matches the offline version of our approach and outperforms the online variants of RotatE. After all data is observed our online method achieves results closest to the best offline method's results.

	Sports Org.	**Religious Entities**
At time $t-1$	St. Louis Blues Orlando Pirates Sheffield Wednesday FC Malaya national football team	Isalm Russian Orthodox church Buddhism United Church of Christ
At time t	Excelsior Rotterdam Seattle Super Sonic	The Mormons Eastern Rite Catholic
	People	**Professions**
At time $t-1$	Marvin Gay Shaquille O'Neal Avril Lavinge Woody Harrleson	Statistician Assoc. football manager Structural Engineer Financial backer
At time t	Elliot Smith Barbara Stanwick	Harpsichordist Child Actor

Table 3.8: Example Clusters discovered in online setting. We show the assignment of new entities to the clusters in the particular time step (below line). Note time is a randomized order of nodes/edges in the knowledge graph and not chronological.

3.6 Proofs and Additional Details

3.6.1 ROTATE Correctly Clusters Strictly Separated Data

Theorem 1 *Let $\boldsymbol{X} = \{x_1, \ldots, x_N\}$ be a dataset and let f is a linkage function such that $\boldsymbol{X}$ is model-based separated by f with respect to a clique structured underlying graph (Assumption 1, Example 4). The hierarchical clustering $\mathcal{T}_i$ constructed via Algorithm 5 has dendrogram purity 1.0.*

Proof. Inductively, assume our current cluster tree $\mathcal{T}_{i-1}$ has dendrogram purity 1.0, and we process a new point x_i belonging to ground truth cluster $C^\star \in \mathcal{C}^\star$. In the first case, assume that $\mathcal{T}_{i-1}$ already contains some members of $C^\star$ that are located in a (pure) subtree. We denote the root of this subtree as $\mathcal{T}_{i-1}[C^\star]$. By the separability assumption, x_i's nearest neighbor must be in $\mathcal{T}_{i-1}[C^\star]$ and if rotations ensue, no node $n \in \mathcal{T}_{i-1}[C^\star]$ can be rotated outside of $\mathcal{T}_{i-1}[C^\star]$. To see why, observe that:

$$f(\mathcal{T}_{i-1}[C^\star], x_i) > f(\mathcal{T}_{i-1}[C^\star], \texttt{aunt}(\mathcal{T}_{i-1}[C^\star])). \tag{3.7}$$

The aunt of the subtree root is by definition from a different ground truth cluster than x_i, $\texttt{aunt}(\mathcal{T}_{i-1}[C^\star]) \not\subseteq C^\star$. After inserting x_i, the subtree root will become $\mathcal{T}_{i-1}[C^\star] \cup \{x_i\}$ and is pure: $\mathcal{T}_{i-1}[C^\star] \cup \{x_i\} \subseteq C^\star$.

In the second case, assume that $\mathcal{T}$ contains no points from cluster $C^\star$. Let v be the nearest neighbor of x_i. By separability, recursive rotations must lift x_i out of the pure subtree $\mathcal{T}[C^\star(v)]$ containing v. This is because the separability assumption tells us that for any $n \in \mathcal{T}[C^\star(v)]$:

$$f(n, x_i) < f(n, \texttt{aunt}(n)). \tag{3.8}$$

This allows $\mathcal{T}_i$ to maintain perfect purity. ■

3.6.2 Rotation Lemma Proof

Lemma 1 (Rotation Lemma) *Given a dataset $\boldsymbol{X}$ and a linkage function f such that $\boldsymbol{X}$ is model-based separated with respect to f and latent graph $G = (\boldsymbol{X}, E)$, let $\mathcal{T}_{i-1}$ be a hierarchical clustering produced by* GRINCH *using linkage function f over the first $i-1$ points, and $\mathcal{T}_i = \text{ROTATE}(x_i, \mathcal{T}_{i-1}, f)$ with $\texttt{lvs}(\mathcal{T}) = \boldsymbol{X}$, all nodes that were strongly connected in $\mathcal{T}_{i-1}$ are strongly connected in $\mathcal{T}_i$, i.e., rotations preserve strong connectivity.*

Proof. Let V be a maximal strongly connected node in $\mathcal{T}_{i-1}$ and assume that x is added as a leaf of V (rotations have not yet been applied). Consider two cases:

(1) there exists an edge between x and some point in V, and

(2) there does not exist an edge between x and any leaf in V.

Case 1: Let $L \subseteq V$ be the points in V to which x is connected. Then x is initially added as a sibling of its nearest neighbor leaf, x', and $x' \in L$ because f separates G according to model-based separation. $\texttt{parent}(x)$ is strongly connected because there exists an edge between x and x'.

The addition of x does not disconnect V or any strongly connected descendant of V. To see why, consider the siblings of the ancestors of x' before the addition of x. Any such sibling that was connected to x', is, after the addition of x, also connected to $\texttt{parent}(x)$ and thus remains strongly connected. Nodes that are not ancestors of x cannot be disconnected and thus, before rotations, strong connectivity is preserved.

Now consider subsequent rotations. By the logic above, x and its sibling, $x' = \texttt{sib}(x)$, are connected. If a rotation succeeds then x and $\texttt{aunt}(x)$ are swapped. So long as $\texttt{aunt}(x)$ and $\texttt{sib}(x)$ form a connected subgraph in G, i.e., $\phi(\texttt{sib}(x), \texttt{aunt}(x)) = \phi(x, \texttt{sib}(x)) = 1$, then the rotation preserves strong connectivity.

The only way for a rotation to disrupt strong connectivity is if x and $\texttt{aunt}(x)$ are swapped, and $\texttt{sib}(x)$ and $\texttt{aunt}(x)$ do not form a connected subgraph in G, i.e., $\phi(x, \texttt{sib}(x)) > \phi(\texttt{sib}(x), \texttt{aunt}(x))$. But, because f separates G, $\phi(x, \texttt{sib}(x)) > \phi(\texttt{sib}(x), \texttt{aunt}(x)) \implies$

$f(x, \texttt{sib}(x)) > f(\texttt{sib}(x), \texttt{aunt}(x))$ and so, in this case, a rotation will not be performed and the procedure terminates.

Case 2: If there does not exist an edge between x and point in V, then after x is made a sibling of some leaf $x'' \in V$, V is no longer strongly connected and so strong connectivity has not been preserved. Since V was strongly connected before the addition of x, there exists an edge between the points in $\texttt{sib}(x)$ and the points in $\texttt{aunt}(x)$. Since f separates G, $f(x, \texttt{sib}(x)) < f(\texttt{sib}(x), \texttt{aunt}(x))$, which triggers the ROTATE subroutine. Rotations proceed with respect to x at least until x is no longer a descendant of V, and thus, V remains strongly connected. Strongly connected nodes that are not descendants of V are unaffected by the rotations and so strong connectivity is preserved. ■

3.6.3 Grafting Lemma 1 Proof

Lemma 2 (Grafting Lemma 1) *Given a dataset $\boldsymbol{X}$ and a linkage function f such that $\boldsymbol{X}$ is model-based separated with respect to f and latent graph $G = (\boldsymbol{X}, E)$, let $\mathcal{T}_i$, which is built with linkage function f on the first i points of $\boldsymbol{X}$, satisfy strong connectivity and completeness, and let V be a node in $\mathcal{T}_i$ such that V is either a maximal strongly connected node or not strongly connected. The tree, $\mathcal{T}_i'$, resulting from the $\texttt{graft}$ operation initiated from V, $\texttt{graft}(V, \mathcal{T}_i, f)$, satisfies strong connectivity and completeness.*

Proof. Let $\mathcal{T}$ be strongly connected and complete. Since V is not a strict subset of any connected component in G, there does not exist a non-empty subset $S \subseteq \boldsymbol{X}_i \backslash V$ such that $S \cup V$ is a connected subgraph in G. For any node V' that is strongly connected but not maximal, there must be an edge connecting V' and $\texttt{sib}(V')$ and $\texttt{sib}(V')$ must be strongly connected, so $f(V', \texttt{sib}(V')) > f(V, V')$. Therefore, an attempt to make any such V' the sibling of V fails.

If V'' is a maximal strongly connected node, an attempt to make V'' the sibling of V may succeed but this does not disconnect any strongly connected subtrees in $\mathcal{T}$. The same is true if V'' is not strongly connected. ■

3.6.4 Grafting Lemma 2 Proof

Lemma 3 (Grafting Lemma 2) *Given a dataset $\boldsymbol{X}$ and a linkage function f such that $\boldsymbol{X}$ is model-based separated with respect to f and latent graph $G = (\boldsymbol{X}, E)$, let $\mathcal{T}_i$, which is built with linkage function f on the points $x_1, \ldots, x_i$ corresponding to graph G_i, satisfy strong connectivity. Let V be strongly connected, but not complete with respect to the first i points, $V \subsetneq (C \cap \{x_1, \ldots, x_i\})$ for some C that is a connected component in G_i. Then a* `graft` *initiated from V returns a node V' such that V' is strongly connected and $V \subsetneq V' \subseteq C$.*

Proof. Since V are a strict subset of the vertices in the connected component, C, there exists a non-empty subset S in $\boldsymbol{X}_i \backslash V$ such that $S \cup V$ constitute the vertices in C. Now define L to be the leaf closest to V according to $f(\cdot, \cdot)$ (line 3 of `graft` (Algorithm 6)):

$$L = \left\{ \underset{x \in X_i,\ x \notin V}{\operatorname{argmax}} \ f(\{x\}, V) \right\}. \tag{3.9}$$

By the fact that V is a strict subset of a connected component C, there must exist an edge between V and L.

If $f(V, L) < f(L, \texttt{sib}(L))$, then there must exist an edge between L and a node in $\texttt{sib}(L)$ and so $\texttt{parent}(L)$ is strongly connected. If $f(V, L) < f(V, \texttt{sib}(V))$, then there must exist an edge between a node in V and a node in $\texttt{sib}(V)$ and so $\texttt{parent}(V)$ is strongly connected. In both of these cases, we do not merge V with L, but instead attempt another merge between two strongly connected nodes, either: $\texttt{parent}(V)$ with L, V with $\texttt{parent}(L)$, or $\texttt{parent}(V)$ with $\texttt{parent}(L)$. As before, the two nodes we are attempting to merge also have an edge between them.

Let V_1 and V_2 be two nodes involved in a merge and let $V_1 \in \texttt{ancs}(V)$ and $V_2 \in \texttt{ancs}(L)$. If at some point

$$f(V_1, V_2) > \max[f(V_1, \texttt{sib}(V_1)), f(V_2, \texttt{sib}(V_2))] \tag{3.10}$$

then V_2 is made a sibling of V_1 and the new parent of V_1 is returned. Since V_1 and V_2 are strongly connected and there exists an edge between the points in V_1 and points in V_2, $\texttt{parent}(V_1)$, which is created by the merge, is strongly connected, and the lemma holds.

If a merge is never performed, the recursion stops when $V_1 = V_2 = \mathsf{lca}(V, L)$. In this case, the `lca`, which we return, is already strongly connected and, by definition, its leaves are a superset of V. ■

3.6.5 Restructure Lemma Proof

Lemma 4 (Restructuring Lemma) *Let $V \in \mathcal{T}_i$ be strongly connected. Let $A \in \texttt{ancs}(V)$ be the deepest connected ancestor of V such that: A is not strongly connected, and all siblings of the nodes on the path from V to A are strongly connected. Then `restruct` on inputs V and A restructures the tree rooted at A so that A satisfies strong connectivity.*

Proof. Let Z be the deepest ancestor of V that is strongly connected with parent $\texttt{parent}(Z)$ that is disconnected. Since $\texttt{parent}(Z)$ is disconnected (but by assumption both Z and $\texttt{sib}(Z)$ are connected), there are no edges between Z and $\texttt{sib}(z)$.

Let A' be a child of A and without loss of generality, $A' \notin \texttt{ancs}(Z)$. Since A is the deepest connected ancestor of Z, there must exist an edge between Z and A'.

When computing the argmax of $f(Z, \cdot)$ in the `restruct` method, a node, Z', that is connected to Z will be returned and then swapped with $\texttt{sib}(Z)$. The new parent of Z is strongly connected because Z and Z' are both strongly connected and there exists an edge between Z and Z'. Any subsequent swap attempted from a disconnected node with a connected ancestor succeeds and produces a new parent that is strongly connected.

Since A is connected and a swap among the descendants of A does not change A, swapping preserves the connectedness of A. Therefore, swaps proceed until the node A is reached at which point A must be strongly connected.

Note that a swap attempt between a strongly connected node and a node to which it is not connected fails, because f separates G. A swap attempt between a connected node and a node to which it is connected succeeds and produces a new parent that is strongly connected. ■

3.6.6 Theorem - GRINCH Correctly Clusters Model-based Separated Data Proof

Theorem 3 *Given a dataset $\boldsymbol{X}$ and a linkage function f such that $\boldsymbol{X}$ is model-based separated with respect to f and latent graph $G = (\boldsymbol{X}, E)$, let $\mathcal{C}^\star$ be the target partition corresponding to the separated data.* GRINCH *builds a hierarchical clustering $\mathcal{T}_{|\boldsymbol{X}|}$ such that $\mathcal{C}^\star$ is a tree consistent partition of $\mathcal{T}$ regardless of the ordering of points in $\boldsymbol{X}$.*

Proof. We show by induction that if GRINCH is used to build a tree, $\mathcal{T}_i$, over vertices, $\boldsymbol{X}_i$, then the connected components of G are a tree consistent partition in $\mathcal{T}_i$. Furthermore, $\mathcal{T}_i$ satisfies strong connectivity.

Clearly, the theorem holds for the base case: a tree with a single node.

Let $\mathcal{T}_{i-1}$ be the tree built on the first $i-1$ points, $\boldsymbol{X}_{i-1}$. Assume the inductive hypothesis: that $\mathcal{T}_{i-1}$ satisfies completeness and strong connectivity. Now vertex x_i arrives.

If there does not exist an edge, in the latent graph G, between x_i and any other vertex in $\boldsymbol{X}_i$, then after rotations, $\mathcal{T}_i'$ satisfies completeness. Since $\forall A \in \texttt{ancs}(x_i)$, A is a not a strict subset of any connected component in G, by Grafting Lemma 1, subsequent `graft` attempts from the ancestors of x_i preserve strong connectivity and completeness and so the theorem holds.

Assume that x_i is connected to some set of leaves $S \subseteq \boldsymbol{X}_{i-1}$. Since $\mathcal{T}_{i-1}$ satisfies strong connectivity, by the Rotation Lemma, after x_i is added and rotations terminate, $\mathcal{T}_i$ satisfies strong connectivity. Note that $\mathcal{T}_i$ may not satisfy completeness if, before the arrival of x_i, the leaves in S formed at least 2 distinct connected subgraphs in G.

After rotations, a series of `graft` attempts are performed. Consider the first `graft` initiated at $\texttt{parent}(x_i)$. By Grafting Lemma 2, the attempt returns a strongly connected

ancestor of x_i whose leaves are a strict superset of $\texttt{parent}(x_i)$. If a `merge` is performed that moves a node V and makes it a sibling of V', then strong connectivity may be violated. However, notice that the only nodes that can be disconnected by such a merge are the node that, prior to the merge, were ancestors of v and also descendants of $A = \mathsf{lca}(V, V')$.

After the merge, A is restructured, and by the Restructuring Lemma, the resulting tree satisfies strong connectivity. Subsequent calls to `graft` proceed from A. Notice that each invocation of `graft` returns a new strongly connected node with a strictly larger number of descendant leaves, until the resulting tree satisfies completeness. Therefore, successive grafting followed by restructuring eventually returns a node whose leaves are a connected component of G. Ultimately, after rotations and grafting, $\mathcal{T}'$ must satisfy completeness and strong connectivity.

■

CHAPTER 4

LEVEL-WISE HIERARCHICAL AGGLOMERATIVE CLUSTERING

A limitation of GRINCH and the incremental algorithms in Chapter 3 is that they can only mildly exploit parallel computation. While this is acceptable in the truly incremental setting where one point arrives at a time, it limits the applicability of these algorithms to massive datasets in the batch setting where all (or a large portion) of the data is provided initially (with additional data potentially arriving incrementally). In this chapter, we describe a class of bottom-up clustering algorithms that interpolates between hierarchical agglomerative clustering (or the reciprocal nearest neighbor algorithm [217]), and Affinity clustering [31]. We show how the algorithm can naturally be extended to a mini-batch incremental algorithm using tree-structure re-arrangements.

4.1 Sub-Cluster Component Algorithm

The pairwise-sequential merging of HAC limits its scalability to large datasets. This method requires a large number of rounds to achieve meaningful and complete hierarchical clusterings. Affinity clustering on the other hand attempts to resolve this using multifurcating tree structures, merging multiple nodes together in a single round. While efficient, this often leads to significant over-merging of nodes, resulting in a loss of quality in the clustering structure.

Like these methods, our proposed algorithm works in a best-first manner: determining which points should belong together in clusters in a sequence of rounds with each round corresponding to a level of a tree structure. The sequence of rounds begins with the decisions

that are "easy to make" (e.g., points that are clearly in the same cluster) and prolongs the later, more difficult decisions until these confident decisions have been well established.

Let $\mathcal{C}^{(i)}$ be the partition produced after round i and the partition at the starting round be $\mathcal{C}^{(0)} = \{\{x\} \mid x \in \boldsymbol{X}\}$. Let a *sub-cluster* refer to a member a partition, i.e. $C \in \mathcal{C}^{(i)}$. Let $\tau_1, \ldots \tau_L$ be a series of L predefined increasing thresholds, given as hyperparameters to the algorithm. To specify the "condition" under which we merge sub-clusters in each round, we define *sub-cluster component* as:

Definition 11. (Sub-cluster Component) *Two sub-clusters $C_j, C_k \in \mathcal{C}$ are defined to be part of the same* sub-cluster component *according to a threshold τ and linkage $f : \mathcal{P}(\boldsymbol{X}) \times \mathcal{P}(\boldsymbol{X}) \to \mathbb{R}$, denoted $\textsc{Ch}_f(C_j, C_k, \tau, \mathcal{C}) = 1$, if there exists a path $P \subseteq \mathcal{C}$ defined as $\{C_j = C_{s_0}, C_{s_1}, C_{s_2}, \ldots C_{s_{R-1}}, C_{s_R} = C_k\}$, where the following two conditions are met:*

1. $f(C_{s_r}, C_{s_{r-1}}) \geq \tau$ *for* $0 \leq r \leq R$, *and*
2. *either* $C_{s_{r-1}} = \operatorname{argmax}_{C \in \mathcal{C}} f(C_{s_r}, C)$ *and/or*
   $C_{s_r} = \operatorname{argmax}_{C \in \mathcal{C}} f(C_{s_{r-1}}, C)$.

Inference at round i works by merging the sub-clusters in round $i-1$ that are in the same *sub-cluster component*. Computationally, the construction of sub-cluster components can be thought of as the connected components of a graph with nodes as the sub-clusters from the previous round and edges between pairs of nodes that are nearest neighbors and have similarity at least τ.

We define, $\textsc{SC}_f(C_j, \mathcal{C}, \tau)$, as the union of all sub-clusters in $\mathcal{C}$ that are within the sub-cluster component of C_j, i.e.,

$$\textsc{SC}_f(C_j, \mathcal{C}, \tau) := \bigcup_{\substack{C \in \mathcal{C}, \\ \textsc{Ch}_f(C_j, C, \tau, \mathcal{C}) = 1}} C \tag{4.1}$$

Algorithm 9 Sub-Cluster Component Alg. (SCC)

Input: $\boldsymbol{X}$: dataset , f: linkage function, $\{\tau_1, \ldots, \tau_L\}$: a set of thresholds in decreasing order
Output: $(\mathcal{C}^{(0)}, \mathcal{C}^{(1)}, \ldots)$: One flat partition per round
$\mathcal{C}^{(0)} \leftarrow \{\{x\} \mid x \in \boldsymbol{X}\}$
$\mathsf{idx} \leftarrow 1$, $\ell \leftarrow 1$
while $\mathsf{idx} < L$ **do**
    Set $\mathsf{SC}_f(C_\ell, \mathcal{C}^{(\ell-1)}, \tau^{(\ell)})$, $\forall C_i \in \mathcal{C}^{(\ell-1)}$, (Eq. 4.1)
    Set $\mathcal{C}^{(i)}$ (Eq. 4.2)
    $\mathsf{idx} \leftarrow \mathsf{idx} + \mathbb{I}[\mathcal{C}^{(\ell)} = \mathcal{C}^{(\ell-1)}]$
    $i \leftarrow \ell + \mathbb{I}[\mathcal{C}^{(\ell)} \neq \mathcal{C}^{(\ell-1)}]$
    $\tau^{(\ell)} \leftarrow \tau_{\mathsf{idx}}$
return $(\mathcal{C}^{(0)}, \ldots, \mathcal{C}^{(\ell-1)})$

Thus, $\mathrm{SC}_f(C_j, \mathcal{C}^{(i-1)}, \tau^{(i)})$ is a new cluster, created by taking a union of all clusters from round $i-1$ that are in the sub-cluster component of C_j. We create the flat partition at round i, $\mathcal{C}^{(i)}$, as the set of all of these newly found clusters:

$$\mathcal{C}^{(i)} := \{\mathrm{SC}_f(C, \mathcal{C}^{(i-1)}, \tau^{(i)}) \mid C \in \mathcal{C}^{(i-1)}\} \tag{4.2}$$

We refer to our algorithm as the **Sub-Cluster Component algorithm (SCC)**. Algorithm 9 gives pseudocode for SCC. We only increment the threshold if no clusters are merged in the previous round i.e. $\mathcal{C}^{(i-1)} = \mathcal{C}^{(i)}$. The sub-cluster component in a particular round can be found efficiently using a connected components algorithm [49]. Any of the rounds can be used as a predicted flat clustering. A hierarchical clustering is given by $\bigcup \mathrm{SCC}(\boldsymbol{X}, f, \{\tau_1, \ldots, \tau_L\})$, the union of the sub-clusters produced by all rounds. Figure 4.1 provides an illustration of the SCC algorithm and the sub-cluster formation.

4.1.1 Practical Considerations: Operating on Sparse Similarity Graph

To speed up computation when using linkages such as average/single/complete, we pre-compute a nearest-neighbor graph over the dataset and only use the similarities defined as edges in this graph when computing the linkage following. We refer to the k-nearest neighbors of a point x as $\mathsf{knn}(x)$ and the ℓ^{th} nearest neighbor of x as x_{knn_ℓ}. We say that

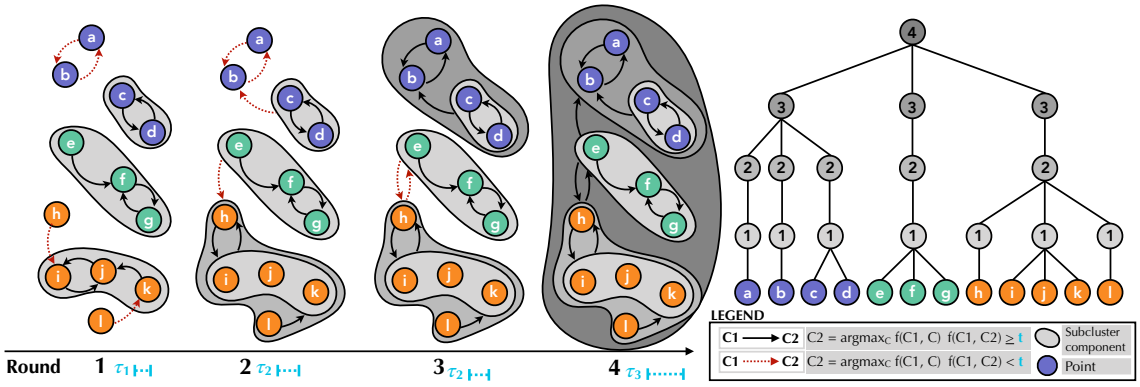


Figure 4.1: **The Sub-Clustering Component Algorithm**. We illustrate SCC on a small dataset. The formation of sub-clusters is shown with black arrows for pairs of points satisfying Def. 11. The direction indicates indicates the nearest neighbor relationship (Def. 11, condition 2). Red edges indicate the nearest neighbor relationships that are above the distance thresholds. The grey circles indicate the sub-cluster components created in that round. Best viewed in color.

two clusters C_j and C_k are connected if $\exists x \in C_j$, such that $C_k \bigcap \mathsf{knn}(x) \neq \emptyset$ or $\exists x \in C_k$, such that $C_j \bigcap \mathsf{knn}(x) \neq \emptyset$. Average linkage can be computed by assuming 0 similarity between points not connected in the nearest neighbor graph:

$$A(C_j, C_k) = \{(x_j, x_k) | x_j \in C_j \wedge x_k \in \mathsf{knn}(x_j) \cap C_k\} \tag{4.3}$$

$$\cup \{(x_j, x_k) | x_k \in C_k \wedge x_j \in \mathsf{knn}(x_k) \cap C_j\} \tag{4.4}$$

$$f(C_j, C_k) = \begin{cases} \frac{1}{|C_j||C_k|} \sum_{x_j, x_k \in A(C_j, C_k)} f(x_j, x_k) & \text{connected} \\ 0 & \text{otherwise.} \end{cases} \tag{4.5}$$

When two clusters are merged, their nearest-neighbor edges can be contracted using the same approach as in [31].

4.1.2 Practical Considerations: Computation of Sub-cluster components

To find the subcluster components, we use efficient, distributed connected components algorithms such as the SV Algorithm [251] or FastSV [301]. We note that in practice only a

small number of rounds of the SV Algorithm are needed especially given the sparse number of edges in the components.

4.1.3 Practical Considerations: Sharding Datasets

We find that simply sharding datasets into coarse grained clusters using algorithms like k-means or DP-means is very effective. We use the shards (which can be overlapping) to restrict the construction of the nearest neighbor graph of SCC. For datasets where the resulting nearest neighbor graph fits on a single machine, we can run the parallel, but not distributed version of SCC. If we need to distribute the nearest neighbor graph, we can use the same DHT-based approach as in Affinity Clustering [31]. This is used in the experiments performed on datasets with $\sim 48M$ points for discovering word senses.

4.2 Mini-Batch SCC

The SCC algorithm described in Section 4.1 operates in the batch setting where all data is given at once. While SCC offers significant speed advantages over GRINCH, the algorithm in Section 4.1 does not support the addition of continuously arriving data. In this section, we describe a level-wise hierarchical clustering method for the online setting. We show that the approach can be usedfor point-at-a-time clustering, as well as in a mini-batch setting.

We propose, **Mini-Batch SCC** (MBSCC), which combines the level-wise structure of SCC for forming non-binary trees with rotation and graft-like tree re-arrangements as in GRINCH. Like SCC, each level of the structure is formed by finding connected components in a thresholded-based, 1-nearest neighbor graph of the nodes in the previous level. Rotations in MBSCC are achieved by using the level-wise thresholds. Grafts are achieved by changing the 1-nearest neighbor relationships in structure of the level. The level-wise structure allows us to naturally support the addition of mini-batches of data added simultaneously, unlike previous work in incremental and online hierarchical clustering.

Algorithm 10 Mini-Batch SCC (MBSCC)

Input: $\boldsymbol{X}_{t-b:t}$: mini-batch of dataset, f: linkage function, $(\mathcal{C}_{t-b}^{(0)}, \ldots, \mathcal{C}_{t-b}^{(L)})$: clustering from previous time step
Output: $(\mathcal{C}_t^{(0)}, \ldots, \mathcal{C}_t^{(L)})$: clustering after observing the mini-batch
Initialize $\mathcal{U}_t^{(0)} \leftarrow \{\{x\} | x \in \boldsymbol{X}_{t:t+b}\}$
Add the k-nearest neighbors of each $\boldsymbol{X}_{t:t+b}$ to $\mathcal{U}_t^{(0)}$.
for i from 0 to L **do**
    Update 1-nearest neighbor for each $C_j^{(i)} \in \mathcal{U}_t^{(i)}$ (Eq. 4.7)
    Update sub-cluster components for each $C_j^{(i)} \in \mathcal{U}_t^{(i)}$ (Eq. 4.8)
    Build next round's nodes to update $\mathcal{U}_t^{(i+1)}$ (Eq. 4.11).
    Build next round's clusters, $\mathcal{C}_t^{(i+1)}$ (Eq. 4.12)
return $(\mathcal{C}_t^{(0)}, \ldots, \mathcal{C}_t^{(L)})$

Recall that each node $C_j^{(i)}$ in level i, needs to know its 1-nearest neighbor in order to efficiently find the sub-cluster component to which it belongs in that level $\text{SC}_f(C_j^{(i)}, \mathcal{C}^{(i)}, \tau^{(i+1)})$. We denote the 1-nearest neighbor of $C_j^{(i)} \in \mathcal{C}_i$ as:

$$C_j^{(i)^{\rightarrow}} := \underset{C \in \mathcal{C}_i}{\text{argmax}}\, f(C_j^{(i)}, C) \tag{4.6}$$

While child/parent relationships can be defined by sub/super-set relations, we also use $\texttt{parent}_t(C_j^{(i)})$ to denote the parent of $C_j^{(i)}$ in level $i+1$ at time step t.

Suppose we have built a level-wise SCC tree structure on the first $t-b$ points of a dataset $\boldsymbol{X}$. We would now like to add the next b points $x_{t-b}, \ldots, x_t$ where b is the mini-batch size. Recall that in previous work like GRINCH, if $b > 1$ the algorithm would simply insert the b points sequentially. At a high level, MBSCC will first determine, from the mini-batch, which leaf nodes need to reconsider their placement in the tree structure. For those leaf nodes and the newly arrived points, the algorithm will reconsider 1-nearest neighbor decisions and the resulting sub-cluster component placements. This will form a new parents for a small subset of the leaves. Then these new parents will then recursively become the new nodes in the next level, again re-configuring sub-cluster components. To again highlight the analogy to GRINCH, the decision of a node refraining from joining any sub-cluster component in a

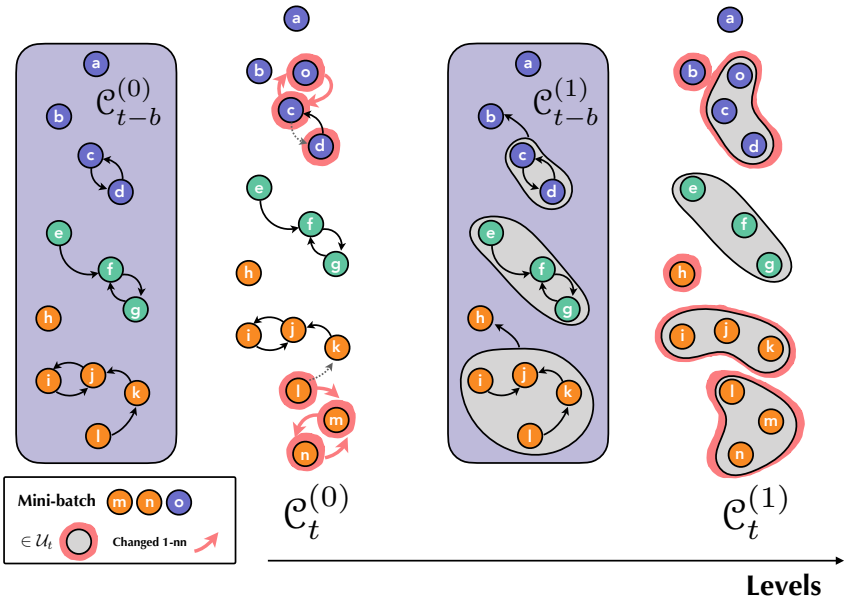


Figure 4.2: **Mini-Batch SCC**. We show the first two levels of the tree structure with the addition of three new datapoints m, n, o. We highlight which nodes are updated in each round in the salmon-red color.

level is like a rotation. The decision of an existing node in the tree to change its 1-nearest neighbor is analogous to a graft.

We will operate level-by-level in a bottom-up fashion. For level i, we will maintain a set of nodes, $\mathcal{U}_t^{(i)}$, that will be updated as the mini-batch is added at time t. MBSCC will update the i^{th} level from time step $t-b$, $\mathcal{C}_{t-b}^{(i)}$ to form $\mathcal{C}_t^{(i)}$. The update has three steps: (1) updating the one-nearest neighbor of each node in $\mathcal{U}_t^{(i)}$, (2) re-forming the sub-cluster components for the nodes in $\mathcal{U}_t^{(i)}$, (3) determining the nodes to update in the next round, $\mathcal{U}_t^{(i+1)}$, based the current round's updated nodes, $\mathcal{U}_t^{(i)}$, and the next round's clusters from the previous time step, $\mathcal{C}_{t-b}^{(i+1)}$. We note that in the leaf level round we define $\mathcal{U}_t^{(0)}$ to be the points in the mini-batch and their k nearest neighbors.

Step (1) is simple and easy to parallelize across nodes in the mini-batch.

$$C_j^{(i)\rightarrow} := \underset{C^{(i)} \in \mathcal{C}^{(i)}}{\operatorname{argmax}} f(C_j^{(i)}, C^{(i)}) \quad \forall {C_j}^{(i)} \in \mathcal{U}_t^{(i)} \tag{4.7}$$

Step (2) will form sub-cluster chains for the marked nodes in $\mathcal{U}_t^{(i)}$. That is, we will potentially re-assign the parent nodes for only those nodes in $\mathcal{U}_t^{(i)}$. Recall that we define, $\text{SC}_f(C_j, \mathcal{U}, \tau)$, as the union of all sub-clusters in $\mathcal{C}$ that are within the sub-cluster component of $C_j^{(i)}$, i.e.,

$$\text{SC}_f(C_j^{(i)}, \mathcal{U}_t^{(i)}, \tau) := \bigcup_{\substack{C^{(i)} \in \mathcal{U}_t^{(i)}, \\ \text{CH}_f(C_j^{(i)}, C^{(i)}, \tau_{i+1}, \mathcal{U}_t^{(i)})=1}} C^{(i)}. \tag{4.8}$$

This process potentially assigns a new parent node for each $C_j^{(i)} \in \mathcal{U}_t^{(i)}$. The parent corresponds to the union of the clusters in the same sub-cluster component. Of course, in this incremental setting, we would like to limit the number of nodes that require an update (i.e., the size of $\mathcal{U}_t^{(i)}$).

Step (3) sets $\mathcal{U}_t^{(i+1)}$. First, we determine the nodes in level $i+1$ that have been deleted, a set which we call $\mathcal{R}_t^{(i+1)}$

$$\mathcal{R}_t^{(i+1)} := \{\texttt{parent}_{t-b}(C) | C \in \mathcal{U}_t^{(i)}\} \setminus \{\texttt{parent}_t(C) | C \in \mathcal{U}_t^{(i)}\}. \tag{4.9}$$

We then set $\mathcal{U}_t^{(i+1)}$ to be ⓐ the new parents of the nodes in $\mathcal{U}_t^{(i)}$ formed by step 2, ⓑ the best neighbors of the previous parents (at time step $t-b$) of the nodes $\mathcal{U}_t^{(i)}$ at time step $t-b$ that are not in $\mathcal{R}_t^{(i+1)}$, ⓒ the siblings of the previous parents (at time step $t-b$) of the nodes $\mathcal{U}_t^{(i)}$ at time step $t-b$ that are not in $\mathcal{R}_t^{(i+1)}$.

$$\begin{aligned} \mathcal{U}_t^{(i+1)} := & \{\texttt{parent}_t(C) | C \in \mathcal{U}_t^{(i)}\} && \text{ⓐ} \\ & \{\texttt{parent}_{t-b}(C)^{\rightarrow} | C \in \mathcal{U}_t^{(i)}\} \setminus \mathcal{R}_t^{(i+1)} && \text{ⓑ} \\ & \{\bigcup \texttt{sib}(\texttt{parent}_{t-b}(C)) | C \in \mathcal{U}_t^{(i)}\} \setminus \mathcal{R}_t^{(i+1)} && \text{ⓒ} \end{aligned} \tag{4.10}$$

Note the importance of using the best neighbors, ⓑ, in addition to the siblings of the previous parents, ⓒ. A parent node might be a singleton in a given level because its nearest neighbor is not within the given threshold. We would like to make sure that that nearest

neighbor node is considered when re-forming the subsequent rounds. Thus we use this edge, which can be thought of storing an "alternative parent" or "DAG parent" for the node invariant to the round threshold, see Figure 4.3.

To complete round i's update, we build $\mathcal{C}_t^{(i+1)}$:

$$A^{(i)} := \{\mathbf{SC}_f(C, \mathcal{U}_t^{(i)}, \tau^{(i+1)}) \mid C \in \mathcal{U}_t^{(i)}\} \tag{4.11}$$

$$\mathcal{C}_t^{(i+1)} := A^{(i)} \cup \{C \setminus \bigcup A^{(i)} | C \in \mathcal{C}^{(i)}\}. \tag{4.12}$$

Notice that there can be nodes whose siblings change that are not in the marked nodes. We need to be careful in assigning parents accordingly (the set difference in Eq. 4.12).

We present pseudocode for MBSCC in Algorithm 10. We visually present key ideas from the Algorithm in Figure 4.2.

4.2.1 Practical Consideration: Early-Stopping Updates

We observe that many mini-batches leave the upper levels of tree structures unchanged. It is often the case that re-building the contracted graph similarities in those levels (Section 4.1.1). Of course, when the next minibatch arrives it might change the lower levels of the tree structure before the upper levels are ever observed. In this way, we can employ lazy updates that only update a level of the tree when it is necessary. We use a early stopping condition that says: when all the parent nodes of a level are the same as the previous mini-batch's decision, break the level-wise update loop. We further note, that based on empirical observations for SCC that indicate that using one level-per-threshold is enough, we simplify the algorithm to do just that.

4.3 Theoretical Analysis

In this section, we provide results for SCC used for both flat and hierarchical clustering. We analyze the separability conditions under which SCC recovers the target clustering and

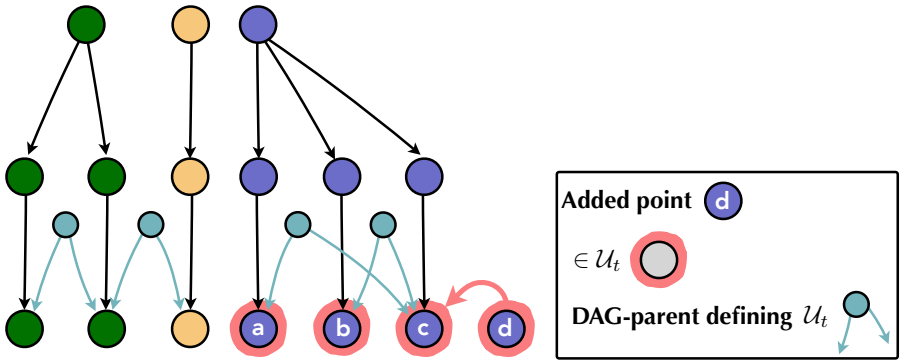


Figure 4.3: **Illustration of using DAG-parents (best neighbors) in MBSCC.** When determining which nodes should be updated in a particular round ($\mathcal{U}_t$) we maintain a structure of alternate parents in a round.

connect these results to the DP-Means objective as well as hierarchical clustering evaluation measures. Lastly, we relate our method to agglomerative clustering and show that in the limit of number of rounds our method will produce the same tree structure as agglomerative clustering.

4.3.1 Recovering the Target Clustering

Each round of SCC produces a flat clustering of a dataset $\boldsymbol{X}$. We will show that if $\boldsymbol{X}$ satisfies the δ-separability assumption (Assumption 2), one of the rounds of SCC will in fact be equal to the target clustering for the dataset, $\mathcal{C}^\star$, corresponding to the separated clusters. To show this, we re-write SCC in terms of distances rather than similarities. This means the thresholds would be monotonically increasing and the linkage function would give a distance rather than a similarity. Formally, we make the statement:

Theorem 4. *Suppose the dataset $\boldsymbol{X}$ satisfies the δ-separability assumption (Assumption 2) with respect to the target clustering $\mathcal{C}^\star = \{C_1^\star, \ldots, C_k^\star\}$ for $\delta \geq \gamma$. $SCC(\boldsymbol{X}, d, \{\tau_0, \ldots, \tau_L\})$ is set of partitions produced by* SCC *(Alg. 9) with $d(\cdot, \cdot)$ as the average distance between points and geometrically increasing thresholds i.e. $\tau_i = 2^i \cdot \tau_0$. The target clustering is equal*

to one of the clustering produced by one round of SCC, $\mathcal{C}^\star \in SCC(\boldsymbol{X}, d, \{\tau_0, \ldots, \tau_L\})$*, where* $\gamma = 6$ *for all metrics and* $\gamma = 30$ *for the* ℓ_2^2 *distance and* $\tau_0 \leq \min_{x,x' \in \boldsymbol{X} \times \boldsymbol{X}} ||x - x'||$.

Intuitively, we prove that, for the aforementioned bounds on within/across cluster distances, a geometric series will necessarily include a threshold that is larger than largest within cluster distance and smaller than the closest across cluster distance between any two sub-clusters. Having such a threshold $\tau^\star$, we will have a round i with a predicted flat clustering, $\mathcal{C}^{(i)}$, equal to the target clustering $\mathcal{C}^\star$, $\mathcal{C}^{(i)} = \mathcal{C}^\star$.

4.3.2 Relation to Nonparametric Clustering

Next, we analyze the performance of our algorithm with respect to nonparametric, flat clustering cost functions. Nonparametric clustering, where the number of clusters is not known *a priori* and must be inferred from the data, is useful for many clustering applications [19, 258]. DP-means [157] is an example of a widely used nonparametric cost function, that is obtained from the small variance asymptotics of Dirichlet Process mixture models.

Definition 12. (DP-Means [157]) *Given a dataset* $\boldsymbol{X}$*, a partition* $\mathcal{C} = \{C_1, \ldots, C_K\}$*, such that cluster* C_l *has center* c_l *and hyperparameter* λ*, the* DP-Means *objective is:*

$$DP(\boldsymbol{X}, \lambda, \mathcal{C}) = \sum_{C_l \in \mathcal{C}} \sum_{x \in C_l} ||x - c_l||^2 + \lambda |\mathcal{C}|. \tag{4.13}$$

Given a dataset $\boldsymbol{X}$ *and hyperparameter* λ*, clustering according to DP-Means seeks to find:* $\operatorname{argmin}_{\mathcal{C}} DP(\boldsymbol{X}, \lambda, \mathcal{C})$.

We show that our proposed algorithm yields a constant factor approximation to DP-Means solution under the data separation Assumption 2. We do so by first showing that SCC contains constant factor approximation solution to Facility Location and then use the relationship between Facility Location and DP-Means [228]. Facility Location problem is defined as:

Definition 13. (Facility Location) *Given a set of clients H as well as facilities F, and a set $\mathbf{f} = \{f_1, \ldots, f_K\}$ of facility opening costs, that is let f_j be the cost of opening facility j and let $e(i, j)$ be the cost of connecting client i to the open facility j. Let $I \subseteq F$ be the set of opened facilities and let $\phi : H \to I$ be the mapping from clients to facilities. The total cost of opening a set of facilities is:*

$$cost(H, F, I, \phi, \mathbf{f}) = \sum_{i \in H} e(i, \phi(i)) + \sum_{i \in I} f_i \tag{4.14}$$

The facility location problem is to solve: $\operatorname{argmin}_{I,\phi} cost(H, F, I, \phi, \mathbf{f})$ *given e, F and $\mathbf{f}$.*

As shown by [228], Facility Location is closely related to DP-Means. In particular, the solution of Facility Location gives a solution to DP-Means:

Definition 14. (DP-Facility [228]) *We define the DP-Facility problem to be the facility location problem where $f_j = \lambda$ for all facilities $j \in F$, $H = F = \boldsymbol{X}$ and e be* squared euclidean distance. *Given a solution $I, \phi = \operatorname{argmin}_{I,\phi} cost(\boldsymbol{X}, \boldsymbol{X}, I, \phi, \lambda)$, we define that $c_\cdot := I$ and $C_k = \{i | \phi(i) = x_k\}$, $K = |I|$ and say that $\mathcal{C} = (C_1, \ldots, C_K)$ is a solution to DP-Means given by the solution of DP-Facility.*

First, we consider δ-separated data in DP-Facility problem and show that the target separated partition gives an optimal solution to DP-Facility:

Theorem 5. *Suppose the dataset $\boldsymbol{X}$ satisfies the δ-separability assumption with respect to clustering $C_1^\star, \ldots, C_k^\star$, then this clustering is an optimal solution to the DP-Facility problem with $\lambda = (\delta - 2) \cdot R$. where $R := \max_{l \in [k]} \max_{x \in C_l^\star} ||x - c_l^*||$.*

We've shown that SCC finds the target clustering for δ-separated data in Theorem 4. We now consider the DP-Means cost of this partition. The next proposition formally relates the DP-Facility problem to an approximate solution to the DP-Means objective.

Proposition 1. [228]. *Let $\boldsymbol{\mu}^\star, Z^\star, K^\star$ be an optimal solution to the DP-Means problem and let $\boldsymbol{\mu}^\dagger, Z^\dagger, K^\dagger$ be the DP-Means solution given by an optimal solution, $I^\dagger, \phi^\dagger$ to the DP-facility location problem. Then,*

$$DP(X, \lambda, Z^\dagger, \boldsymbol{\mu}^\dagger, K^\dagger) \leq 2 \cdot DP(X, \lambda, Z^*, \boldsymbol{\mu}^*, K^*) \tag{4.15}$$

Finally, we can analyze the quality of the solutions found by SCC on δ-separated data, showing that it is a constant factor approximation.

Corollary 6. *Suppose the dataset $\boldsymbol{X}$ satisfies the δ-separability assumption (Assumption 2) with respect to the target clustering $\mathcal{C}^\star = \{C_1^\star, \ldots, C_k^\star\}$ for $\delta \geq \gamma$. $SCC(\boldsymbol{X}, d, \{\tau_0, \ldots, \tau_L\})$ is set of partitions produced by* SCC *with $d(\cdot, \cdot)$ as the average distance between points and geometrically increasing thresholds i.e. $\tau_i = 2^i \cdot \tau_0$. $SCC(\boldsymbol{X}, d, \{\tau_0, \ldots, \tau_L\})$ contains the optimal solution to DP-Facility problem with $\lambda = (\delta - 2) \cdot R$ where R is defined in Theorem 5 and finds a solution that is a 2-approximation to the DP-Means objective with the same λ.*

Proof. Theorem 4 and 5, when the data satisfies the δ-separation assumption, then SCC contains the optimal solution to DP-facility problem. Using Proposition 1, we see that this solution is within 2 factor of the DP-means solution. ■

4.3.3 Hierarchical Clustering Analysis

The sequence of partitions produced by SCC forms a hierarchical clustering of the dataset $\boldsymbol{X}$. More precisely, the union of the partitions returned by SCC is a hierarchical clustering, $\bigcup \text{SCC}(\boldsymbol{X}, \{\tau_1, \ldots, \tau_L\}, d)$.

Theorem 4 shows that SCC will recover the target clustering for data satisfying Assumption 2. We relate this to the metric used to evaluate hierarchical clusterings given a labeled flat partition, dendrogram purity [146, 170]. We show our proposed algorithm will have perfect (equal to 1) dendrogram purity (Equation 2.4) on well separated data. This follows naturally from the above theorem.

Corollary 7. *Suppose the dataset $\boldsymbol{X}$ satisfies the δ-separability assumption with respect to the target clustering $\mathcal{C}^\star = \{C_1^\star, \ldots, C_k^\star\}$ for $\delta \geq \gamma$. $SCC(\boldsymbol{X}, d, \{\tau_0, \ldots, \tau_L\})$ is set of partitions produced by* SCC *with $d(\cdot, \cdot)$ as the average distance between points and geometrically increasing thresholds i.e. $\tau_i = 2^i \cdot \tau_0$. Perfect dendrogram purity is achieved by* SCC*, $\textit{DendrogramPurity}(\bigcup \text{SCC}(\boldsymbol{X}, d, \{\tau_1, \ldots, \tau_L\})) = 1$, where $\gamma = 6$ for all metrics and $\gamma = 30$ for the ℓ_2^2 distance and $\tau_0 \leq \min_{x, x' \in \boldsymbol{X}^2} ||x - x'||$.*

Proof. There exists a round i equal to the target clustering. Each descendant of the target clusters will therefore have purity 1. Each pair of points will have a least common ancestor as one of the descendants of the nodes in round i or a node in round i. And so the purity of the least common ancestor for every pair of same-cluster points will have purity 1, leading to dendrogram purity to be 1. ■

4.3.4 Relationship to Agglomerative Clustering

The bottom-up nature of our algorithm is reminiscent of hierarchical agglomerative clustering, another round-based algorithm in which each round merges the two subtrees with minimal distance according to a linkage function, $f : \mathcal{P}(\boldsymbol{X}) \times \mathcal{P}(\boldsymbol{X}) \to \mathbb{R}$. We make the following statement about SCC, returning to using similarities rather than distances:

Proposition 2. *Let $f : \mathcal{P}(\boldsymbol{X}) \times \mathcal{P}(\boldsymbol{X}) \to \mathbb{R}$ be a linkage function that is symmetric and injective, $C_1, C_2, C_3, C_4 \subset \boldsymbol{X}$, $f(C_1, C_2) = f(C_3, C_4) \iff (C_1 = C_3 \wedge C_2 = C_4) \vee (C_1 = C_4 \wedge C_2 = C_3)$ (i.e., the linkage between each pair of nodes is unique). Let $\mathcal{T}$ be the tree formed by HAC (Algorithm 1) and let f also satisfy reducibility [55], $\forall C, C', C'' \in \mathcal{T}$, $f(C, C') \geq \max\{f(C, C''), f(C', C'')\} \implies \max\{f(C, C''), f(C', C'')\} \geq f(C \cup C', C'')$ then there exists a sequence of threshold $t_1, \ldots, t_r$ such that the tree formed by $\bigcup \text{SCC}(\boldsymbol{X}, f, \{\tau_1, \ldots, \tau_r\})$ is the same as $\mathcal{T}$.*

Algorithm 1 reveals the relationship between HAC and SCC. We set the thresholds used in SCC to only merge one pair per round, producing the same sequence of mergers present in HAC to take place in SCC.

4.3.5 Worst-Case Time Analysis

For a given round, let $O(T)$ be the time required to build a 1-nearest neighbor graph over the sub-clusters of a round. Cover Trees [38] allow this to be done in $O(c^{12} N \log N)$ for N elements where c is the expansion constant. In the worst case, SCC requires $2 * (N - 1)$ rounds (merging one pair of elements per round, multiplicative factor due to determining when to advance idx in Algorithm 9). This leads to a running time $O(NT)$. Further, we find that using a fixed number of rounds and simply advancing the threshold in each round experimentally works well.

4.4 Batch Setting Experiments

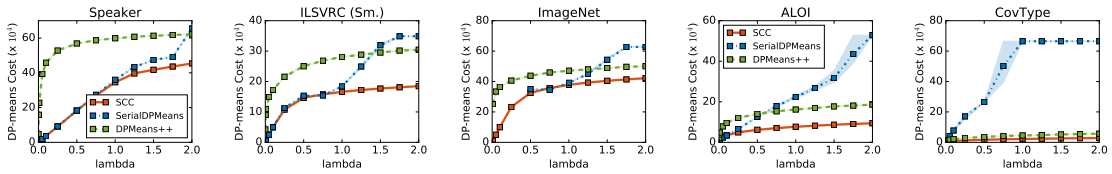


Figure 4.4: **DP-Means Cost** for the solutions found with a variety of methods for a range of λ values from close to 0 to 2. SCC produces lower cost solutions for different values of λ as compared to the other methods.

Our analysis showed that SCC can not only recover optimal partitions of the data, but also higher quality hierarchies. In this section, we want to experimentally validate these claims on a variety of publicly available clustering benchmarks. Specifically we want to demonstrate that:

- SCC recovers more accurate hierarchical clustering than state-of-the-art methods (Section 4.4.1).
- SCC produces high quality flat partitions of the data (Section 4.4.2).
- SCC produces high quality solution to the DP-means objective (Section 4.4.3).

Datasets: We evaluate SCC on the following publicly available clustering benchmark datasets as in [170] (Table 4.1): **CovType** - forest cover types; **Speaker** - i-vector speaker

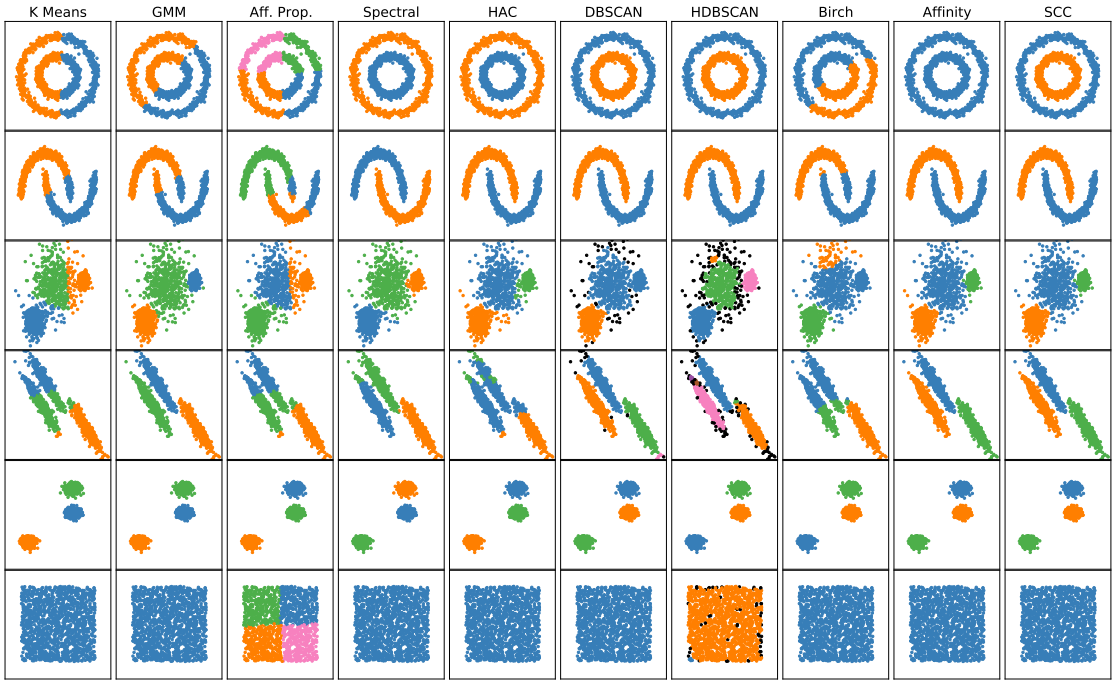


Figure 4.5: **Comparison of Clustering Algorithms on Small Datasets**. We extend the comparison of clustering algorithms in low-dimensional data from scikit-learn [229] to include SCC. We observe that DBSCAN, HAC, and SCC are the top performing methods.

recordings, ground truth clusters refer each unique speaker [132]; **ALOI** - 3D rendering of objects, ground truth clusters refer to each object type [123]; **ILSVRC (Sm.) (50K subset)** and **ILSVRC (Lg.) (1.2M Images)** images from the ImageNet ILSVRC 2012 dataset [242] with vector representations of images from the last layer of the Inception neural network; **ImageNet** features of the same kinds of images extended to all 17K classes present in ImageNet. In all experimental settings, we use the average linkage function with either dot products as similarity or ℓ_2^2 as distance.

To speed up computation, we pre-compute a nearest-neighbor graph over the dataset and only use the distances defined as edges in this graph when computing distances between clusters as in Section 4.1.1.

	CovType	ILSVRC (Sm.)	ALOI	Speaker	ImageNet	ILSVRC (Lg.)
$\lvert\mathcal{C}^\star\rvert$	7	1000	1000	4958	17K	1000
$\lvert\boldsymbol{X}\rvert$	500K	50K	108K	36.5K	100K	1.3M
dim.	54	2048	128	6388	2048	2048
gHHC	0.444	0.381	0.462	-	0.020	0.367
GRINCH	0.430	0.557	0.504	0.48	0.065	-
PERCH	**0.448**	0.531	0.445	0.372	0.065	0.207
BIRCH	0.44	0.26	0.32	0.22	0.03	0.11
HKMeans	0.44	0.12	0.44	0.12	0.02	0.11
HDBSCAN	**0.473**	0.414	**0.599**	0.396	-	-
Affinity	0.433	0.587	0.478	0.424	0.055	0.601
SCC	0.433	**0.622**	0.575	**0.510**	**0.072**	**0.606**

Table 4.1: **Dendrogram Purity** results on benchmark datasets. gHHC did not produce meaningful results on Speaker and GRINCH did not scale to ILSVRC (Lg.). We find that Affinity and SCC outperform the baselines with SCC giving best performance on all datasets except one.

4.4.1 Hierarchical Clustering

We first analyze the performance of SCC compared to state-of-the-art hierarchical clustering algorithms: **Affinity** [31] (Algorithm 3); **Grinch** [210] (Algorithm 8); **Perch** [170] - an online hierarchical clustering algorithm that creates trees one point at a time by adding points next to their nearest neighbor and perform local tree re-arrangements in the form of rotations; **gHHC** [211] - a gradient-based hierarchical clustering method that uses a continuous tree representation in the unit ball. Here we present SCC results using dot product similarities and geometrically decreasing thresholds and a comparison to ℓ_2^2 in Table 4.5.

Each dataset has ground truth flat clusters and we evaluate using dendrogram purity (Equation 2.4) as in previous work. As seen in Table 4.2, we observe that SCC achieves the highest dendrogram purity on all datasets except CovType. Notably, both SCC and Affinity clustering scale much better to the largest 1.2M image dataset, with both methods having no degradation in performance on this dataset from the 50K subset to the larger 1.2M point dataset.

Dataset	SCC	Affinity	K-Means	Perch
CovType	**0.536**	**0.536**	0.245	0.230
ILSVRC (Sm.)	0.609	**0.632**	0.605	0.543
ALOI	**0.567**	0.439	0.408	0.442
Speaker	**0.493**	0.299	0.322	0.318
Imagenet	**0.076**	0.055	0.056	0.062
ILSVRC (Lg.)	0.602	**0.641**	0.562	0.257

Table 4.2: **Pairwise F1** when selecting a flat clustering with ground truth # of clusters.

	ALOI	**ILSVRC (Sm.)**	**ILSVRC (Lg.)**
HDBSCAN	1635	7902.90	DNF
GRINCH	385.113	836.748	DNF
AFFINITY	29.01 + 0.834	261.831 + 0.298	849.846 + 9.886
SCC	29.01 + 2.79	261.831 + 4.29	849.846 + 65.621

Table 4.3: **Running Time (seconds) of Top Performing Methods**. Each run on machine using 24 2.40GHz CPUs. Grinch and HDBSCAN did not finish on largest dataset in 10 hours. Affinity and SCC report Sparse Graph Construction Time + Algorithm Execution Time for given graph. DNF = Did not finish in 10hours.

We report running times in Table 4.3. The time required by SCC and Affinity is dominated by the construction of the sparse nearest neighbor graph. We use ScaNN to build the nearest neighbor graph [137]. We use the publicly available implementations of HDBSCAN.

We also compare SCC's performance to HAC and found that SCC matches the performance of HAC while being much more scalable (Section 4.4.5). We analyse different threshold schedules in Section 4.4.4.

4.4.2 Flat Clustering

We first compare a wide variety of algorithms on synthetic datasets in Figure 4.5. Then, we empirically evaluate SCC against state-of-the-art approaches in terms of pairwise F1 and the DP-Means objective. We use the same experimental setting as in previous works [170]. In this experiment, we use the ground truth number of clusters when selecting a flat

clustering. We select the round of SCC which produced the closest number of clusters to the ground truth and report the flat clustering performance of that round. We do the same for baseline methods. We evaluate the quality of the flat clusterings using the pairwise F1 metric [170, 199].

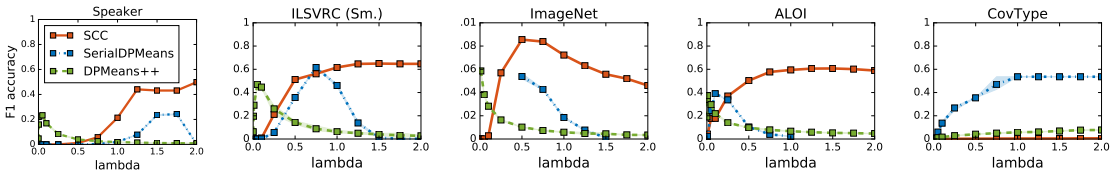


Figure 4.6: **Pairwise F1 Evaluation**. As each algorithm depends on λ in a different way, the settings of λ resulting in the best performance might differ between methods. We plot the performance of each method for each value of λ. When considering the best F1 achieved by each method for some value of λ, SCC is the top performing method on 4 of 5 datasets.

Table 4.2 gives the F1 performance for each method. We observe that both SCC and Affinity outperform the previous state-of-the-art results and that the best performing approach varies from dataset to dataset. We also report the best F1 achieved in any round of our algorithm and Affinity, which is the next best performing method (see also Section 4.4.6). We observe in Table 4.6 that the best F1 is better from our approach than from Affinity, highlighting that our tree structures potentially contain more high quality alternative clusterings.

4.4.3 Summary of DP-Means Experiments

We measure the quality of the clustering discovered by our algorithm in terms of the DP-Means objective following our analysis (Corollary 6).

We compare SCC to the following state-of-the-art algorithms for obtaining solutions to the DP-means objective: **SerialDPMeans**, the classic iterative optimization algorithm for DP-Means [53, 157, 176, 228], in which data point is added to a cluster if it is within λ of that cluster center and otherwise starts a new cluster (for large datasets we use its distributed variant **OCC** [228]); **DPMeans**++ [23] an initialization-only method which performs a K-Means++ [270] style sampling procedure. For each method, we record the assignment

of points to clusters given by inference. We use this assignment of points to flat clusters to produce a DP-Means cost.

Figure 4.4 shows for each dataset the DP-Means objective as a function of the value of the parameter λ. Figure 4.6 shows the corresponding F1 performance for different values of λ. Each method uses normalized ℓ_2^2 distance as the dissimilarity measure. We report the min/max/average performance over multiple runs of the SerialDPMeans and DPMeans++ algorithms with different random seeds. A more detailed explanation is provided in Section 4.4.8.

We observe that for each value of λ, SCC achieves the lowest DP-Means cost. We hypothesize that our improvement is due foremost to having an algorithm that discovers optimal clustering independently of λ and our algorithm's ability to explore multiple alternative partitions in the tree. For each method, if we consider the best F1 value achieved by the algorithm, SCC is usually one of the best performing methods. In Section 4.4.8, we analyse SCC performance with different number of rounds, provide running times comparison and large scale experiments results.

4.4.4 Hyperparameter Analysis

Affinity clustering does not require any hyper-parameters. For Grinch, the results / settings from the original paper are used. For gHHC, we performed a grid search over the number of internal nodes used, learning rate, and child/parent regularization terms.

SCC Thresholds We experiment with two kinds of thresholds, one based on ℓ_2^2 as shown in our theoretical results and one based on using similarities (dot products/cosine similarities) between the points. We find that the two perform comparably on 3 out of the 5 datasets and that dot products work better on ALOI and Speaker (Table 4.5). Dot products give state-of-the-art results on all datasets and ℓ_2^2 gives state-of-the-art results on all except Speaker. We use the normalized ℓ_2^2 distance which is bounded in the range [0,4] and a range of [0,1] for dot products. For distances, we use a geometric progression with base $(\frac{M}{m})^{\frac{1}{L}}$ and

	CovType	ILSVRC (Sm.)	ALOI	Speaker	ImageNet	ILSVRC (Lg.)
Exponential	0.433	0.622	0.575	0.510	0.0722	0.606
Linear	0.433	0.641	0.572	0.491	0.0798	0.591

Table 4.4: **Comparison of Round Threshold Schedules**. We run SCC with two settings of round thresholds, exponentially decaying and linear schedules. We use 30 rounds in each case. We report the dendrogram purity of each and observe that the performance of the exponential is typically better.

Metric	Fixed # Rounds	CovType	ILSVRC (Sm.)	ALOI	Speaker	ImageNet
ℓ_2^2	Y	0.437	0.617	0.537	0.446	0.076
ℓ_2^2	N	0.443	0.626	0.554	0.455	0.077
$x_i^T x_j$	Y	0.438	0.631	0.586	0.524	0.074
$x_i^T x_j$	N	0.438	0.632	0.588	0.524	0.075

Table 4.5: **Distance/Similarity Metric Comparison & Fixed Number of Rounds**. We observe that ℓ_2^2 and $x_i^T x_j$ give comparable results on 3 out of 5 datasets and $x_i^T x_j$ improves results on two datasets. We observe that using a fixed number of rounds with one round per threshold does not impact performance.

$\tau_0 = m$, so our thresholds are $m \cdot (\frac{M}{m})^{\frac{i}{L}}$ for $i = 1$ to L where m is the minimum allowed pairwise distance and M is the maximum allowable pairwise distance. As $M/m > 1$, this meets the criteria of the theorems we proved. For similarities, we use the comparable geometrically increasing progression. We compare this to use a linear progression of the thresholds in Table 4.4. We find both schedules work quite well with linear working slightly better on the two ILSVRC datasets. Further, we compare using a fixed number of rounds, incrementing the threshold index after each round as an approximation of our method. We find that the results are nearly identical regardless of whether the threshold is incremented or not (Table 4.5).

In Table 4.4, we compare the performance of our approach using exponentially decaying round parameters to using the linear schedule of rounds. We find that on most datasets the performance is typically quite similar, but find some improvements using the exponential scheme.

4.4.5 Comparison to HAC

We sample datasets of varying sizes from a Dirichlet Process Mixture Model. Figure 4.7 reports the running time and dendrogram purity as a function of the dataset size. We report SCC results with 200 rounds. We observe that while the time complexity of HAC grows quadratically, SCC remains much more constant. Despite being much more efficient than HAC we observe that SCC produces trees with comparable dendrogram purity.

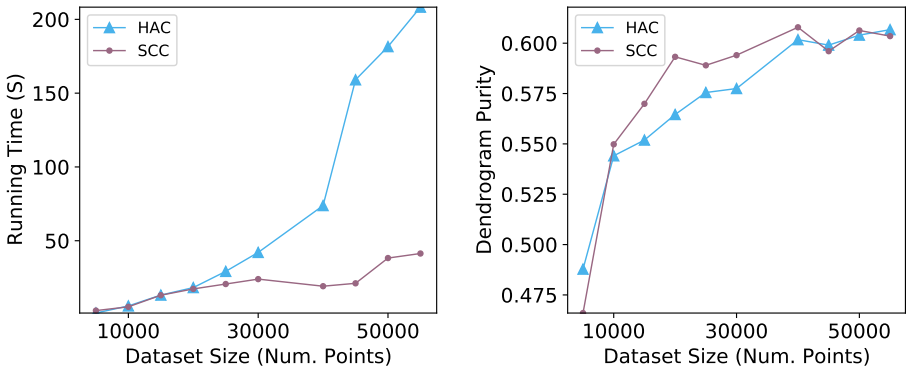


Figure 4.7: **Comparison to HAC**: We report running time and Dendrogram Purity of SCC compared to HAC, on a synthetic dataset.

4.4.6 Flat clustering Comparison to Affinity

In the hierarchical clustering and flat clustering experiments, we observe that SCC and affinity clustering were typically the best performing methods. We observe that the flat clustering produced when using the ground truth number of clusters is not always the best in terms of F1. We report in Table 4.6 the results of the best F1 of any flat partition produced in a round of Affinity and SCC.

4.4.7 Comparison to Robust Hierarchical Clustering (RHC)

RHC [29] has stronger theoretical guarantees than SCC. RHC achieves these stronger theoretical guarantees by using a complex linkage function, requiring more time per linkage function execution. In Table 4.7, we compare the best dendrogram purity achieved by

	CovType	ILSVRC (Sm.)	ALOI	Speaker	ImageNet	ILSVRC (Lg.)
Affinity	0.536	0.632	0.465	0.3141	0.055	0.641
SCC	*0.536*	*0.654*	*0.605*	*0.526*	*0.081*	*0.664*

Table 4.6: **Best F1** The best F1 achieved by the methods for any number of clusters. We observe that the best F1 achieved by SCC is consistently best.

	RHC	SCC
Iris	**0.955**	0.926
Wine	0.944	**0.975**

Table 4.7: **Comparison to RHC** We report the best dendrogram purity achieved across various hyperparameter settings of each method.

SCC and RHC on the Iris and Wine datasets using a grid search over each method's hyperparameters ($\alpha + \nu$ for RHC, number of nearest neighbors and rounds for SCC). We use the publicly available MATLAB implementation of RHC. We find on these small datasets that SCC achieves competitive dendrogram purities despite using a simpler linkage function.

4.4.8 Detailed DP-means Experiments

For each method, we recorded the assignment of points to clusters given by inference. We then use this assignment of points to flat clusters to produce a DP-Means cost. Note that this means that while methods like exemplar based clustering method would produce a data point as the cluster center, we instead replace this representative with the empirical mean of the cluster points. This strictly improves the DP-Means objective score (see Proposition 1). Each method uses normalized ℓ_2^2 as the dissimilarity measure.

Figure 4.4 shows for each dataset the DP-Means objective as a function of the value of the parameter λ. Each method uses normalized ℓ_2^2 distance as the dissimilarity measure. We use the following range of lambda values 0.001, 0.005, 0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 1.0,

1.25, 1.5, 1.75, 2.0. We report the min/max/average performance over multiple runs of the SerialDPMeans and DPMeans++ algorithms with different random seeds.

Noticeably, each algorithm depends on the value of λ in a different way. SerialDPMeans relies directly on λ to determine membership of clusters. This means that for values of λ greater than the maximum distance between points the algorithms will put all points into the same cluster and return the clustering with the entire dataset with a single cluster. DPMeans++ has similar behavior, but behaves more globally with respect to the λ as its stopping condition depends on the sum of the distance of every point to cluster assignment. Our method, SCC, on the other hand constructs a series of candidate solutions to DP-Means independent of λ and then selects amongst these clusterings, the one which optimizes the DP-Means objective for a given value of λ.

4.4.8.0.1 F1 results Each dataset has a ground truth flat partition, defined for purposes of using the datasets for classification. We evaluate the ability of clustering algorithms to recover this ground truth flat partition. We measure the quality of a predicted partition by the pairwise precision/recall/F1 clustering metric [199, 202].

As each algorithm uses the value of λ differently, the value of λ that results in the best F1 score on the dataset could be quite different for each method and for each dataset. We run each method with the same values of λ as in the DP-Means objective evaluation and record the F1 score of the clustering for each. We show this in Figure 4.6. For each method, if we consider the best F1 value achieved by the algorithm, SCC is usually one of the best performing methods. We do observe that both DPMeans++ and SCC perform poorly on the CovType dataset. This is because both predict many more clusters than are needed for the dataset (See §4.4.8.4).

4.4.8.1 Comparison to LowrankALBCD

In this experiment, we compare the performance of our method and LowrankALBCD [292]. The parameter settings described by the authors in their paper as well as the code

provided by the authors use values of lambda at the scale of $\lambda = 0.01 \cdot N$. Our experiments (following those in [292]) use normalized ℓ_2^2 distance. Any value of λ greater than the maximum pairwise distance which in the case of ℓ_2^2 is 4, will result in methods such as SerialDPMeans and OCC-DP-Means necessarily giving solutions of every data point being placed in the same cluster. We evaluated LowrankALBCD on CovType, ALOI, Speaker, and ILSVRC (Small). We tried to use the code provided by the authors of LowrankALBCD to solutions with small values of λ in the range 0 to 4, for large numbers of iterations we found the code either required more than 100GB of RAM or took longer than 10 hours. And so, we instead compare our method and LowrankALBCD for larger values of lambda, specifically the authors suggestion of $0.01N$. We observe that on SCC and LowrankALBCD perform similarly on several of the datasets and LowrankALBCD performs better on CovType. However, we notice that for all datasets except CovType, these values of λ produce unreasonably few clusters. For instance, ALOI and ILSVRC (Sm.) have 1000 ground truth clusters and these values of λ are producing less than 30 clusters. This leads to extremely low pairwise F1 scores. Similarly, the Speaker dataset has around 5000 ground clusters, yet these values of λ produce less than 10 clusters. As reported in the main body of the paper, SCC is able to achieve high F1 scores for lower values of λ when there are fine-grained clusters.

4.4.8.2 Large scale experiments

In this experiment, we scale each of the methods for DP-Means to the largest dataset, a dataset of 1.2 million imagenet images (ILSVRC (Lg.)). We run SerialDPMeans using its parallel implementation Optimistic Concurrency Control [228]. We run SCC, SerialDPMeans/OCC, DPMeans++ for the same values of λ used in the other experiments. We additionally run SCC and DPMeans++ for larger values of λ as one might expect is necessary for a larger dataset.

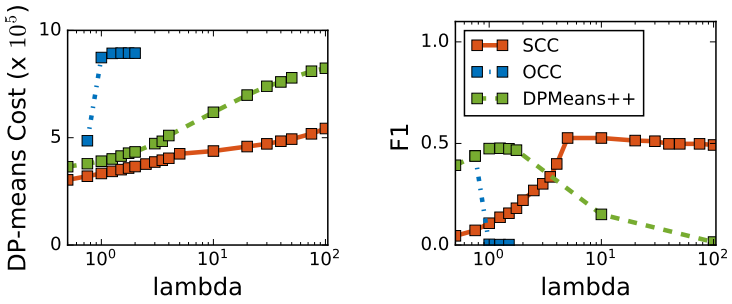


Figure 4.8: DP-means and F1 accuracy for ILSVRC (Lg.) dataset

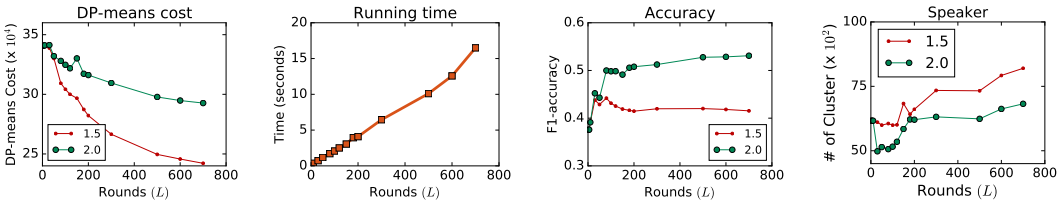


Figure 4.9: **Number of Rounds** The impact on number of rounds (L) in SCC algorithm, on DP-mean cost, running time, F1, and number of clusters on Speaker dataset. For DP-means and F1, we report for $\lambda = \{1.5, 2\}$.

SerialDPMeans/OCC can only be run with $\lambda < 4.0$ since $\lambda > 4.0$ will place all points in the same cluster (the maximum ℓ_2^2 distance between points is 4). We find that running SerialDPMeans/OCC for 50 iterations as in the other datasets is prohibitively time consuming. We find that it takes longer than 10 hours to run more than 2 iterations of OCC, for lambda values less than 0.75. We report the results after the method had finished 2 iterations in the time limit. We observe that the lambda value of 0.75 gives a reasonable F1 score for SerialDPMeans/OCC. We observe that SCC achieves higher F1 scores when the value of λ is larger, as having too small a λ value creates too many clusters. We observe that SCC produces lower costs than DPMeans++ and achieves a higher F1 value for particular values of λ.

4.4.8.3 Number of Rounds - DP-Means

Rounds (L). After fixing the maximum value of the threshold and the mimimum, the number of rounds (L), determine how τ_i changes. To analyse the performance of SCC algorithm with number of rounds we increased L from 2 to 700. In Figure 4.10, we plot DP-means cost, the total distance of points from assigned clusters, the number of clusters discovered, the time taken and the F-1 accuracy vs number of rounds, for two values of $\lambda = \{1.5, 2\}$.

As seen in Figure 4.10, across datasets, the DP-means cost decreases with number of rounds but then the decrease tapers off around 100 - 200 rounds. Moreover, the number of clusters mostly increases with the number of rounds, decreasing the distance of points to the cluster centers. As expected the number of clusters found using higher value of λ (2.0), is always lower than the number of clusters found using lower value of λ (1.5). The time taken has a linear dependence on the number of rounds. The F1-accuracy numbers also increase with the number of rounds and are somewhat stable beyond 100-200 rounds. All this suggests that using 100-200 rounds is ideal for real world scenario.

4.4.8.4 CovType Performance

CovType is a dataset with 500K points belonging to just 7 different clusters. Because of the large number of points, we observe that for small values of λ, SCC produces far too many clusters. SerialDPMeans does not have the same problem as λ directly acts as a distance threshold giving cluster membership. We observe that for larger values of λ the F1 performance of SCC can be improved. However, we do not find a comparison of F1 score particularly meaningful as we observed the highest F1 value (0.536) when all points are placed in a single cluster.

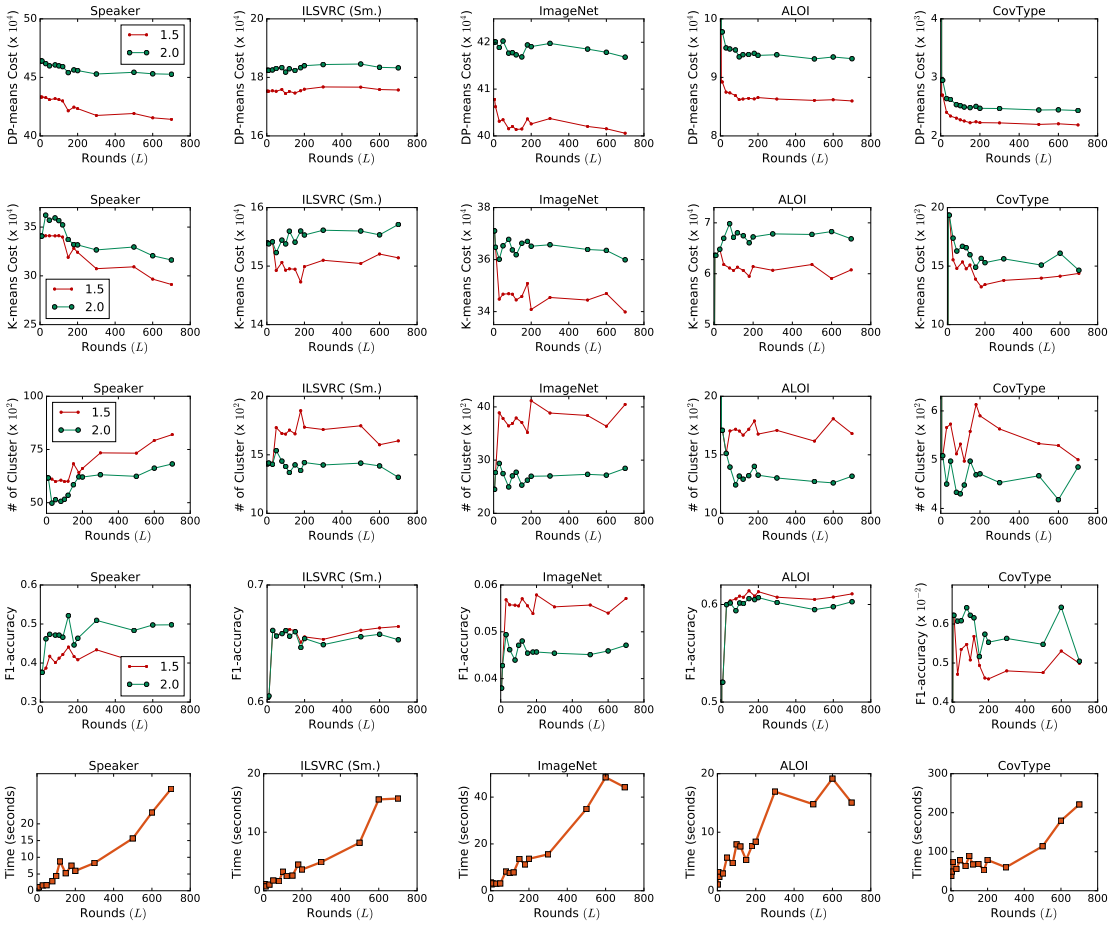


Figure 4.10: **Ablation Study on Number of Rounds Used**. The impact on the number of rounds (L) in SCC algorithm, on DP-mean cost, distance of points from cluster center (K-means cost), number of clusters, F1 and running time for the different datasets. We report numbers for $\lambda = \{1.5, 2\}$. Note that the running time for all λ is the same for SCC, as shown in the last row.

4.5 Use Cases & Applications

In this section, we review four uses cases of SCC with both qualitative and quantitative evaluation.

4.5.1 Word Sense Discovery

We would like to automatically discover the senses of words. For instance, we would hope to find that *leg* can refer to the limb of a human/animal, the support of a table, a part of a journey, etc.

Cluster Summary	Example Members
Query Word: Coach	
sports	footballer, player, captain, leaguer
TV	judges, television, competition, participants, elimination
transit	wagon, carriage, drawing
rail transit	train, railroad, railway
business (n.)	manager, boss, managerial, administration
business (v.)	managed, guided, controlled
train (v.)	practice, trained, hone
Query Word: Wing	
aircraft	nose, tail, tailplane, fin, flap, fuselage
politics	left, right, centrist, faction
hospital	ward, unit, beds, icu, theatre
sports	halfback, tailback, flanker
fly (v.)	fly, flying, gliding
buildings	annex, addition, extension
birds	bird, owl, turkey

Table 4.8: **Example Word Sense Clusters**. Discovered from 48M token embeddings from Wiki103. We show that SCC discovers meaningful senses for words such as *coach* and *wing*.

We build a collection of ~ 48 million word token representations from WikiText-103 [204]. The work token representations that we use are from BERT [99]. We discover senses by clustering all of the word tokens together. The idea is that clusters will form to correspond to a synset of words that have similar contextual meaning. We perform sharding to reduce the complexity of nearest neighbor search. We use ScaNN [137] to perform nearest neighbor search. The largest chunk takes about 5 hours to run on a single machine with 24 CPUs. We find that SCC produces meaningful clusters as shown in Table 4.8

4.5.2 Sentence Clustering

We would like to perform analysis on a dataset where the similarity function is non-metric. We select a corpus of sentences, IBM Debater Thematic Clustering Dataset [107]. We use a sentence-similarity cross encoder [99] to measure the relatedness between sen-

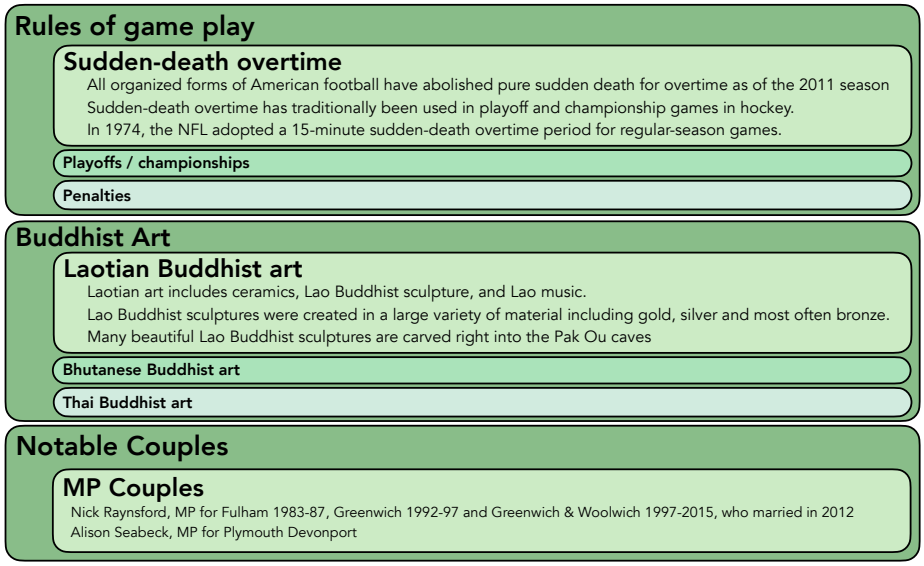


Figure 4.11: **Fine-Grained Sentence Clusters**. Discovered sentence clusters and hierarchy from IBM Debater Dataset.

tences. This does not follow the triangle inequality. We find meaningful flat clusters as well as hierarchical ones, which we illustrate in Figure 4.11.

4.5.3 Biomedical Entity Discovery

Recent work[1] has considered how to automatically discover emerging entities and concepts from biomedical research papers. The approach forms an embedded representation of candidate entity mentions using BioSentVec [75]. These representations are then clustered and the resulting clusters are considered as candidate entities to be added to a knowledge-base such as UMLS. We display example entities and concept hierarchies in Figure 4.12.

4.5.4 Web-Scale Data

We investigate the use of SCC for clustering web queries. We run our proposed clustering approach and Affinity clustering on a dataset of a random sample of 30 billion queries. To the best of our knowledge this is one of the largest evaluations of any clustering

[1] `https://github.com/chanzuckerberg/concept_discovery`

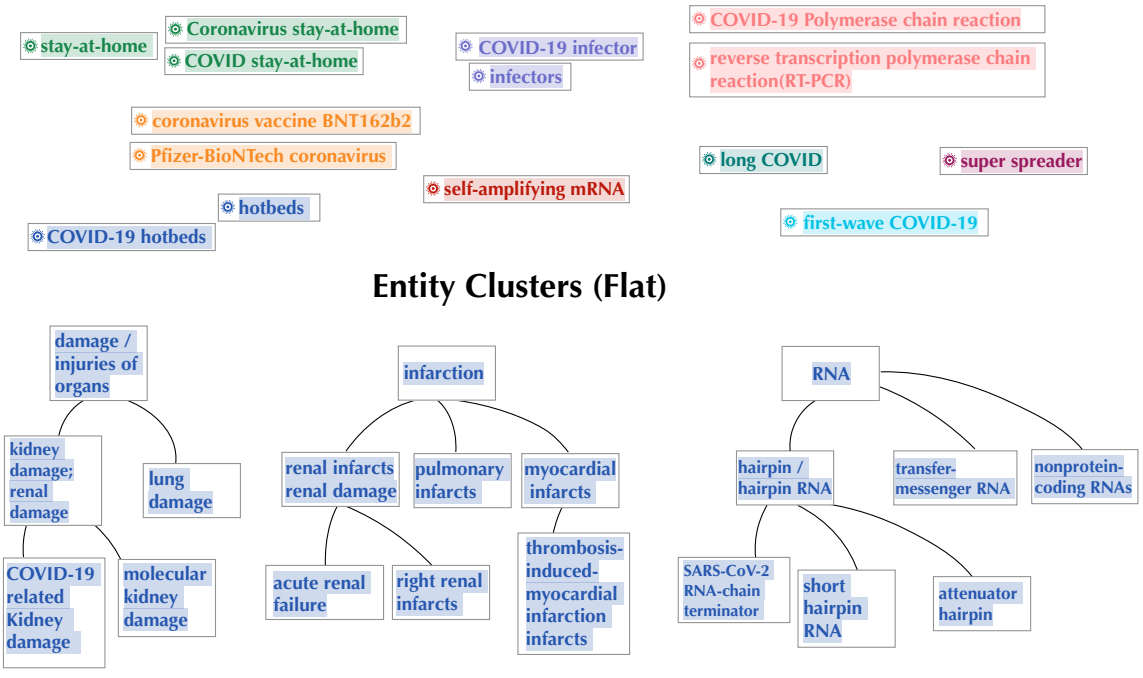


Figure 4.12: **Example BioMedical Concepts**. We present examples of the concepts discovered by our system from PubMed papers related to COVID19. Note that the flat concepts are concepts, which at the time of the experiments, were not in UMLS. We show only partial segments from the discovered hierarchies.

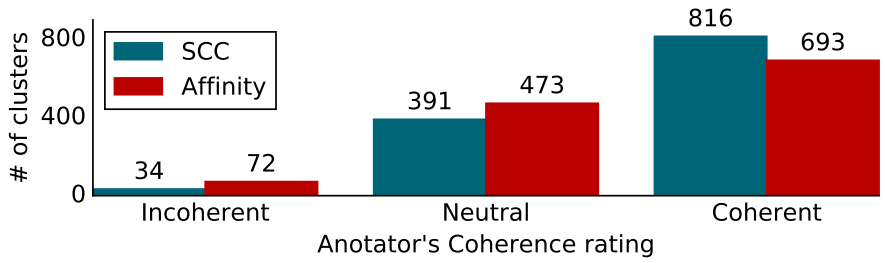


Figure 4.13: Human evaluation of clusters generated by SCC and Affinity

algorithm. Due to the massive size of the data, we limit our evaluation to the two most highly performing methods, SCC and Affinity clustering. Distance computation between queries during algorithm execution are sped-up using hashing techniques to avoid the N^2 pairwise dissimilarity bottleneck (for both SCC and Affinity). Queries are represented using a set of proprietary features comprising lexical and behavioral signals among others. We extract manually a fine-grained level of flat clusterings and compared the clustering quality of the flat clusterings discovered by both algorithms.

Human Evaluation: To evaluate the quality of flat clusters discovered by SCC, we conducted an empirical evaluation with human annotators. We asked them to rate ~ 1200 randomly sampled clusters from -1 (incoherent) to +1 (coherent). For example, a annotator might receive a head query of `home improvement` and a tail query of `lowes near me` from the same cluster. The annotator then rates each of these pairs from -1 (incoherent) to +1 (coherent). We report the aggregated results in Figure 4.13. We found that the annotators labeled 6.0% of Affinity clustering's clusters and only 2.7% of SCC clusters as incoherent. The annotators labeled 55.8% of Affinity's clusters and 65.7% of SCC clusters as coherent. We hope this demonstrates the cogency of the clusters found by SCC.

Qualitative Evaluation: In Table 4.9, we show the clusters discovered by both SCC and Affinity that contain the query *Green Velvet* (the house/techno music artist). We observe that SCC's cluster is considerably more precise and on topic. Table 4.10, shows additional examples. We find the clusters to be precise and coherent. The algorithm discovers a

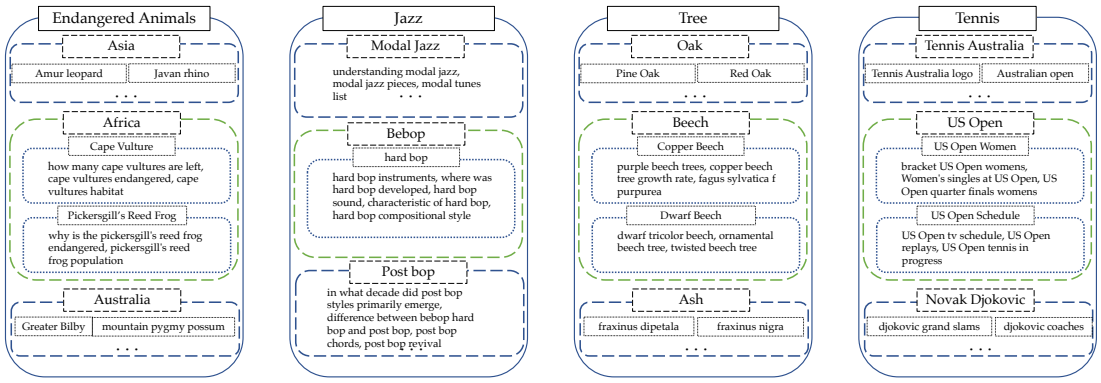


Figure 4.14: Hierarchy inferred on 30 billion user queries using SCC. We represent the hierarchy using rectangular boxes. The root with header is represented by the outer rectangular box (solid line). The second level of the hierarchy, with header, is shown in dashed (− −) rectangle withing the outer box. Finally the third level from the root is shown as the inner most dotted (...) rectangle. Plain text within each of the dotted rectangle are the top queries that belong to that cluster. For example, ENDANGERED ANIMALS, is the root of the left most hierarchy, ENDANGERED ANIMALS IN AFRICA is a sub-cluster and PICKERGILL'S REED FROG is the lowest level cluster containing queries such as WHY IS THE PICKERGILL'S REED FROG ENDANGERED.

clustering related to tennis strategies, which contains meaningful queries such as *baseline tactics* and *tennis strategy angles*. We investigate the hierarchical structure in Figure 4.14. We discover meaningful clusters at multiple granularities with the tree structure indicating meaningful relationships between topics. For instance, we discover subgenres of *jazz* such as *bebop* and *modal jazz*. We further discover that *hard bop* is a subgenre of *bebop*.

Affinity Clustering	**SCC**
green velvet live	green velvet mix
green velvet talking	dj green velvet
kestra financial businesswire	tomorrowland green velvet
tomorrow land green velvet	green velvet 2015
dj green hair	green velvet music
rinvelt & david kestra businesswire	green velvet dj set

Table 4.9: SCC and Affinity Clustering for clusters corresponding to *green velvet* on the 30 billion query dataset.

Tea Recipes	Tennis Strategy	Electric Piano
tea drinks	playing strategies in tennis	digital piano price
tea recipes	understanding tennis tactics	electric piano sale
tea drinks to make at home	tennis strategy names	electric piano prices
fancy tea recipes	baseline tactics	yamaha electric piano small
black tea flavors and recipes	tennis strategy angles	best digital piano ebay

Table 4.10: Example Fine-grained Query Clusters Discovered by SCC

	ALOI	Speaker	ILSVRC12 50k
GRINCH	0.531	**0.525**	0.604
OHAC	0.494	0.453	0.595
FishDBC	0.576	0.306	0.361
MBSCC	**0.591**	0.487	**0.617**

Table 4.11: Comparison of Incremental Methods on Clustering Benchmarks - Dendrogram Purity

4.6 Incremental Setting Experiments

4.6.1 Dendrogram Purity on Clustering Benchmarks

In Table 4.11, we compare MBSCC to other online clustering algorithms including Grinch, OHAC [203], a recently proposed online approximation to agglomerative clustering that deletes all ancestors of the nearest neighbor of a newly observed point and re-clusters those ancestors with HAC, FishDBC [97], an incremental variant of HDBSCAN, and GRINCH. Observe MBSCC is most effective on two of the three benchmarks.

4.6.2 Incremental Running Time

In Figure 4.15, we report the running time of brute force nearest neighbor search (from Faiss [161]) to MBSCC using a hierarchical navigable small world index [161] to build the nearest neighbor graph in an incremental fashion. We observe that can we can observe significant speedups using the mini-batch setting.

4.6.3 Analysis of Batch Size

We would like to understand how performance changes with different mini-batch sizes. We observed speed ups in Figure 4.15. We use exact nearest neighbor search and report just the MBSCC update on the sparsified graph (to minimize variation across runs). The complexity of each round of the incremental algorithm depends on the size of $\mathcal{U}_t^{(i)}$, the number of nodes that will be updated. The overall time complexity will be a function of that cost along with the number of mini-batches that are used.

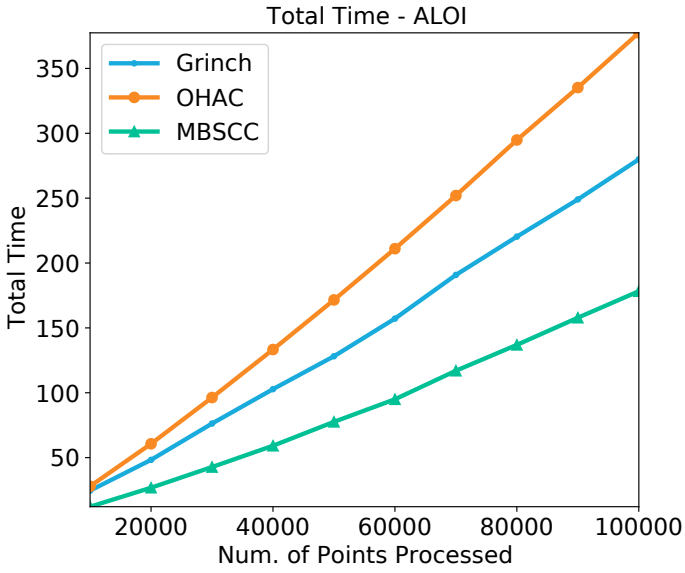


Figure 4.15: **Incremental Algorithm Running Times**. Comparison of the per-point MBSCC to GRINCH and OHAC on the ALOI dataset. MBSCC is both faster and more accurate than GRINCH.

4.6.4 Discovering Evolving Topics in Patent Data

We collect a corpus of around 6M US patents from PatentsView [2]. We train a Sent2Vec embedding [226] to train title representations for the patents following [75]. We apply MBSCC on the collection with mini-batch size of 500K. This takes about 90 minutes on a

[2] patentsview.org

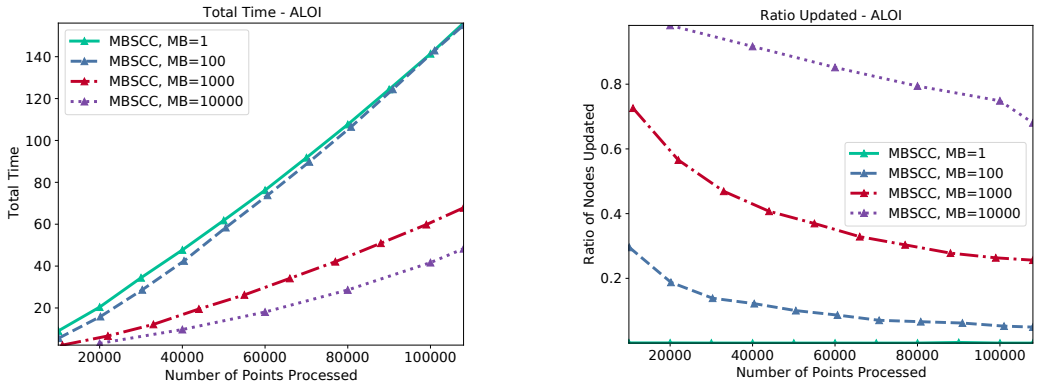


Figure 4.16: **Mini-batch Analysis**. We compare a variety of mini-batch sizes of MBSCC on the ALOI dataset. Left: We show the time to cluster ALOI. Right: We show the amount of the structure that is updated. We find that the number of updates is roughly constant with respect to the batch / dataset size ratio. We find that dendrogram purity increases from about 0.58 with batch size of 1 to 0.62 with any larger batch size.

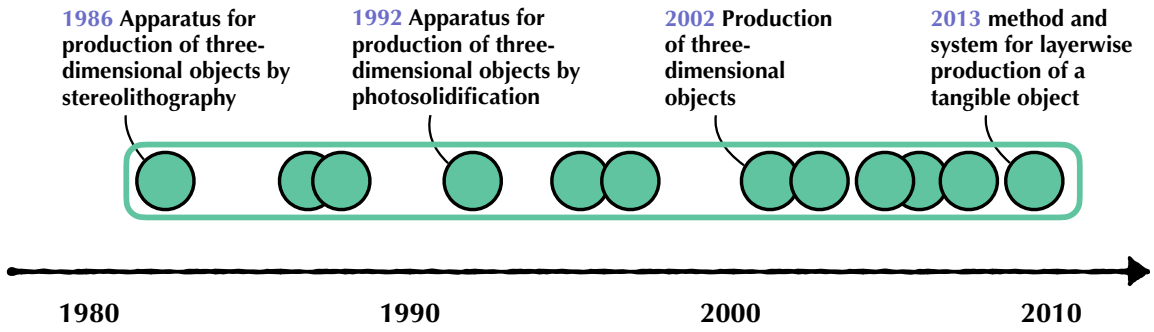
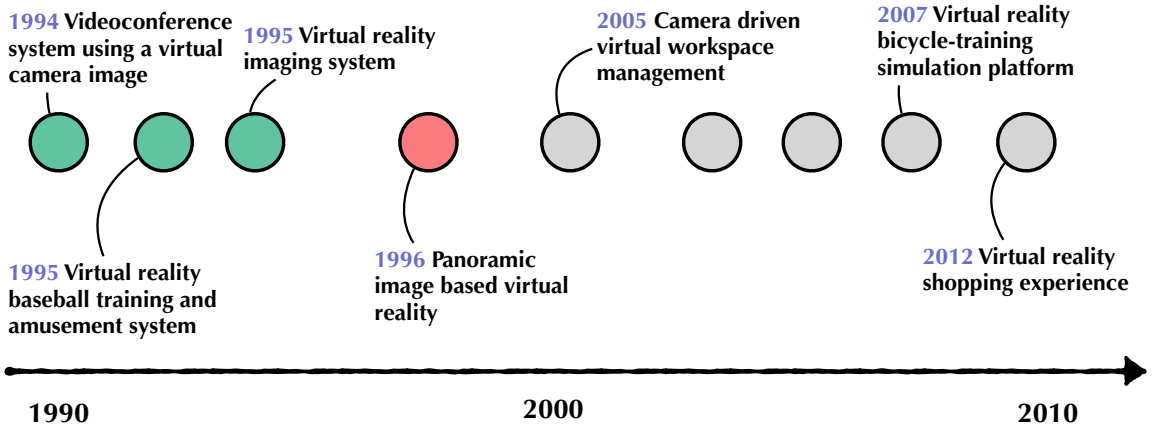
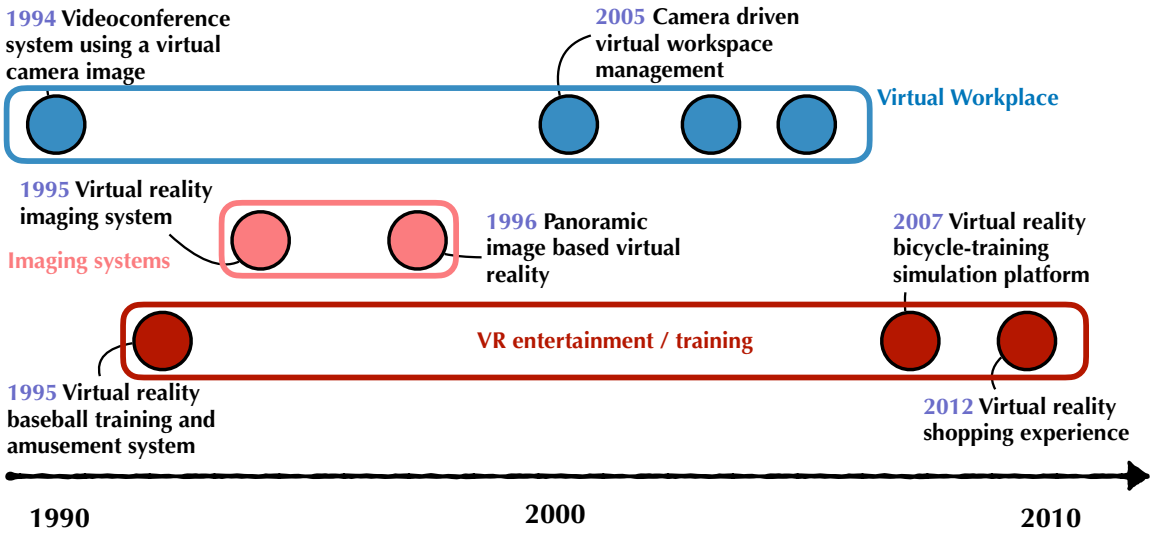


Figure 4.17: **Discovering Emerging work on 3D Printing Patents** We show how the *3D printing* topic grows over time with minibatch SCC.

machine with 12 CPUs. We show two examples of dynamic topic clusters. In Figure 4.17, we show how the *3D printing* topic continues to grow as more patents are granted. In Figure 4.18, we show how the initial clusters related to *virtual reality* change over time. We see that the initial cluster is split into three more fine-grained clusters all of which are related to virtual reality. These clusters include: *virtual workplace*, *imaging systems*, *VR entertainment*. We note that this evolution shows MBSCC's flexibility to effectively reconsider past decisions in the presence of new data.



(a) We present the state of clusters until the year 2000. The grey nodes represent patents observed in future. The green/red clusters represent two clusters of *Virtual Reality* based patents.



(b) We present the state of clusters until the year 2012. Notice how the green cluster above has now been split into three sub-topics related to *Virtual Reality*.

Figure 4.18: **Changing Topics in Virtual Reality Patents**. We are interested to see how *Virtual Reality* related topics are changing as more patents are observed. We find that MBSCC performs rearrangements of past clusters in the presence of new data.

4.7 Proofs and Additional Details

4.7.1 Proof of Proposition 1

Proposition 1 [228] *Let $\boldsymbol{\mu}^*, Z^*, K^*$ be an optimal solution to the DP-means problem and let $\boldsymbol{\mu}^\dagger, Z^\dagger, K^\dagger$ be the DP-Means solution given by an optimal solution, $I^\dagger, \phi^\dagger$ to the DP-facility location problem. Then,*

$$DP(X, \lambda, Z^\dagger, \boldsymbol{\mu}^\dagger, K^\dagger) \leq 2 \cdot DP(X, \lambda, Z^*, \boldsymbol{\mu}^*, K^*). \tag{4.16}$$

Proof. From [228], presented for completeness. It is folklore that this holds for the K-means objective:

$$\min_{Z, \boldsymbol{\mu} \subseteq X} \sum_{i=0}^{N} \sum_{j=0}^{K} z_{i,j} ||\mathbf{x}_i - \boldsymbol{\mu}_j||^2 \leq 2 \min_{Z, \boldsymbol{\mu} \in \mathbb{R}^{N \times d}} \sum_{i=0}^{N} \sum_{j=0}^{K} z_{i,j} ||\mathbf{x}_i - \boldsymbol{\mu}_j||^2 \tag{4.17}$$

And so the K-means result for the optimal K^*.

$$\begin{aligned} DP(X, \lambda, Z^\dagger, \boldsymbol{\mu}^\dagger, K^\dagger) &\leq \min_{Z, \boldsymbol{\mu} \subseteq X} \left\{ \sum_{i=0}^{N} \sum_{j=0}^{K^*} z_{i,j} ||\mathbf{x}_i - \boldsymbol{\mu}_j||^2 \right\} + K^* \lambda \qquad (4.18) \\ &\leq 2 \min_{Z, \boldsymbol{\mu} \in \mathbb{R}^{N \times d}} \sum_{i=0}^{N} \sum_{j=0}^{K^*} z_{i,j} ||\mathbf{x}_i - \boldsymbol{\mu}_j||^2 + K^* \lambda \leq 2DP(X, \lambda, Z^*, \boldsymbol{\mu}^*, K^*) \end{aligned}$$

Recall that $Z^\dagger, \boldsymbol{\mu}^\dagger, K^\dagger$ is an optimal solution and so is at most as expensive as any other solution to the DP facility problem (the first line above). ■

4.7.2 Proof of Theorem 4

Theorem 4. *Suppose the dataset $\boldsymbol{X}$ satisfies the δ-separability assumption with respect to the clustering $C_1^*, \ldots, C_k^*$ for $\delta \geq \gamma$ then the set of partitions produced by* SCC*-algorithm with geometrically increasing thresholds i.e. $\tau_i = 2^i \cdot \tau_0$ contains the optimal partition $C_1^*, \ldots, C_k^*$ where $\gamma = 6$ for all metrics and $\gamma = 30$ for the ℓ_2^2 distance.*

Proof. Let c_i be the center for cluster C_i. Recall the assumption of δ-separability (Assumption 2) in which the maximum distance from any point to its true center is defined as $R := \max_{i\in[k]} \max_{x\in C_i^*} ||x - c_i^*||$. We make the additional assumption that the threshold of the first round, τ_0, is less than R, i.e., $\tau_0 < R$.

First, we give a proof for general metrics and then consider ℓ_2^2. The algorithm begins with $\mathcal{C}^{(0)}$ set to be the shattered partition, with each data point in its own cluster, $\mathcal{C}^{(0)} = \{\{x\}|x \in \boldsymbol{X}\}$.

We want to show that some round, $r^\star$ with threshold τ_r produces the ground truth partition, $\mathcal{C}^{(r^\star)} = \mathcal{C}^* = \{C_1^*, \ldots, C_k^*\}$. We will show by induction that for each round prior to $r' \leq r^\star$ that the clustering $\mathcal{C}^{(r')}$ is *pure*, i.e. $\forall C \in \mathcal{C}^{(r')}$, $\exists C^* \in \mathcal{C}^*$ $C \subseteq C^*$ (equality will be for round $r^\star$). We will show that the round $r^\star$ with $\mathcal{C}^{(r^\star)} = \mathcal{C}^*$ must exist.

Assume that for round $r^\star - 1$, we have *pure* sub-clusters in $\mathcal{C}^{(r^\star-1)}$ such that $X, X' \subseteq C_i^*$, which are disjoint, $X \cap X' = \emptyset$, and $Y \subseteq C_j^*$ for $i \neq j$ ($\mathcal{C}^{(0)}$ by definition has pure sub-clusters). We want to show: every such X and X' must form a sub-cluster component without any such Y. In this way, we ensure that C_i^* exists as a pure cluster in some round.

Recall that we use c to refer to the center of cluster C. Using the triangle inequality we have that:

$$||c_i^* - c_j^*|| \leq \frac{1}{|X||Y|} \sum_{x\in X} \sum_{y\in Y} ||c_i^* - x|| + ||x - y|| + ||y - c_j^*|| \tag{4.19}$$

Re-arranging terms we have:

$$||c_i^* - c_j^*|| \leq \frac{1}{|X|} \sum_{x\in X} ||c_i^* - x|| + \frac{1}{|X||Y|} \sum_{x\in X} \sum_{y\in Y} ||x - y|| + \frac{1}{|Y|} \sum_{y\in Y} ||y - c_j^*|| \tag{4.20}$$

With further re-arrangements,

$$||c_i^* - c_j^*|| - \frac{1}{|X|} \sum_{x\in X} ||c_i^* - x|| - \frac{1}{|Y|} \sum_{y\in Y} ||y - c_j^*|| \leq \frac{1}{|X||Y|} \sum_{x\in X} \sum_{y\in Y} ||x - y|| \tag{4.21}$$

Since $||c_i^* - c_j^*|| \geq \delta \cdot R$, where (see Assumption 2), $R := \max_{i \in [k]} \max_{x \in C_i^*} ||x - c_i^*||$:

$$(\delta - 2) \cdot R \leq ||c_i^* - c_j^*|| - \frac{1}{|X|} \sum_{x \in X} ||c_i^* - x|| - \frac{1}{|Y|} \sum_{y \in Y} ||y - c_j^*|| \leq \frac{1}{|X||Y|} \sum_{x \in X} \sum_{y \in Y} ||x - y|| \tag{4.22}$$

However, we have that for $X \subset C_i^*$ and $X' \subset C_i^*$:

$$\frac{1}{|X||X'|} \sum_{x \in X} \sum_{x \in X'} ||x - x'|| \leq \frac{1}{|X||X'|} \sum_{x \in X} \sum_{x' \in X'} ||c_i^* - x|| + ||c_i^* - x'|| \leq 2R. \tag{4.23}$$

For $\delta \geq 6$, this indicates that there exists a τ_r, such that $2R \leq \tau_r \leq 4R$ for which X and X' would form a sub-cluster component without Y. Since we use a geometric sequence of $\tau_1, \tau_2, \ldots$, we know that τ_r will exist since it is between $2R$ and the doubling of $2R$. X and X' are any sub-clusters of any ground truth cluster C_i^*. The result above indicates that at any round before the one using τ_r that takes as input pure sub-clusters will produce pure sub-clusters as no sub-clusters with points belonging to different ground truth clusters will be merged. Moreover, the existence of τ_r indicates that the partition given by sub-cluster component from a round using τ_r , will contain every ground truth cluster in $\mathcal{C}^*$. In particular, the last round that uses τ_r will be the $r^\star$ to do this, i.e., $\mathcal{C}^{(r^\star)} = \mathcal{C}^\star$. Observe that the separation condition requires that within cluster distances for any two subsets will be less than τ_r and so sub-clusters will continue to be merged together until each ground truth cluster is formed by the last round using τ_r.

Now let's consider the case for ℓ_2^2. We update our definition of R for ℓ_2^2, $R := \max_{i \in [k]} \max_{x \in C_i^*} \|x - C_i^*\|_2^2$. Again, the algorithm begins with $\mathcal{C}^{(0)}$ set to be the shattered partition, with each data point in its own cluster, $\mathcal{C}^{(0)} = \{\{x\} | x \in \boldsymbol{X}\}$. Again, we want to show that for some round, $r^\star$ with threshold τ_r produces the ground truth partition, $\mathcal{C}^{(r^\star)} = \mathcal{C}^* = \{C_1^*, \ldots, C_k^*\}$. As before, we will show by induction that for each round prior to $r' \leq r^\star$ that the clustering $\mathcal{C}^{(r')}$ is *pure*, i.e. $\forall C \in \mathcal{C}^{(r')}$, $\exists C^* \in \mathcal{C}^*$ $C \subseteq C^*$ (equality will be for round $r^\star$). We will show that the round r with $\mathcal{C}^{(r^\star)} = \mathcal{C}^*$ must exist.

As before, we assume that for rounds before and including τ_r, we have *pure* sub-clusters such that $X, X' \subseteq C_i^*$, which are disjoint, $X \cap X' = \emptyset$, and $Y \subseteq C_j^*$ for $i \neq j$ ($\mathcal{C}^{(0)}$ by definition has pure sub-clusters). We want to show: every such X and X' must form a sub-cluster component without any such Y. In this way, we ensure that C_i^* exists as a pure cluster in some round.

Using the relaxed triangle inequality [136] for ℓ_2^2, we have:

$$\|c_i^* - c_j^*\|_2^2 \leq 3\left(\frac{1}{|X||Y|}\sum_{x\in X}\sum_{y\in Y}\|c_i^* - x\|_2^2 + \|x-y\|_2^2 + \|y - c_j^*\|_2^2\right) \tag{4.24}$$

$$\|c_i^* - c_j^*\|_2^2 \leq 3\left(\frac{1}{|X|}\sum_{x\in X}\|c_i^* - x\|_2^2 + \frac{1}{|X||Y|}\sum_{x\in X}\sum_{y\in Y}\|x-y\|_2^2 + \frac{1}{|Y|}\sum_{y\in Y}\|y - c_j^*\|_2^2\right) \tag{4.25}$$

Re-arranging the above:

$$\frac{1}{3}\|c_i^* - c_j^*\|_2^2 - \frac{1}{|X|}\sum_{x\in X}\|c_i^* - x\|_2^2 - \frac{1}{|Y|}\sum_{y\in Y}\|y - c_j^*\|_2^2 \leq \frac{1}{|X||Y|}\sum_{x\in X}\sum_{y\in Y}\|x-y\|_2^2 \tag{4.26}$$

Since $\|c_i^* - c_j^*\|_2^2 \geq \delta \cdot R$ and by the definition of R,

$$(\frac{1}{3}\delta - 2)\cdot R \leq \frac{1}{3}\|c_i^* - c_j^*\|_2^2 - \frac{1}{|X|}\sum_{x\in X}\|c_i^* - x\|_2^2 - \frac{1}{|Y|}\sum_{y\in Y}\|y - c_j^*\|_2^2 \leq \frac{1}{|X||Y|}\sum_{x\in X}\sum_{y\in Y}\|x-y\|_2^2 \tag{4.27}$$

However, for any two subclusters $X, X' \subset C_i^*$ then we know that:

$$\frac{1}{|X||X'|}\sum_{x\in X}\sum_{x\in X'}\|x-x'\|_2^2 \leq 2\left(\frac{1}{|X|}\sum_{x\in X}\|c_i^* - x\|_2^2 + \leq \frac{1}{|X'|}\sum_{x\in X'}\|c_i^* - x'\|_2^2\right) \leq 4R. \tag{4.28}$$

For $\delta \geq 30$, we know that there exists a $4R \leq \tau_r \leq 8R$ for which X and X' would form a sub-cluster component without Y. Since we use a geometric sequence of $\tau_1, \tau_2, \ldots$, we

know that τ_r will exist since it is between $4R$ and the doubling of $4R$. X and X' are any sub-clusters of any ground truth cluster C_i^*. For the same reasons as the metric case, the result above indicates that at the round $r^\star - 1$ we will only have pure sub-clusters as no sub-clusters with points belonging to different ground truth clusters will be merged. This result indicates that the partition given by sub-cluster component from round $r^\star$, $\mathcal{C}^{(r^\star)}$ will contain every ground truth cluster in $\mathcal{C}^*$.

■

4.7.3 Proof of Theorem 5

Theorem 5 *Suppose the dataset $\boldsymbol{X}$ satisfies the δ-separability assumption with respect to clustering $C_1^\star, \ldots, C_k^\star$, then this clustering is an optimal solution to the DP-Facility problem with $\lambda = (\delta - 2) \cdot R$. where $R := \max_{l \in [k]} \max_{x \in C_l^\star} ||x - c_l^*||$.*

Proof. To prove this, we use linear programming duality. To show that this clustering is an optimal solution to the DP-Facility problem, we will use linear programming duality. In particular, we will exhibit a feasible dual, α, to the linear programming relaxation of the DP-Facility Problem, whose cost is the same as the clustering $\{C_1^*, \ldots, C_k^*\}$. From linear programming duality, we know that the following set of relations are true : $\textsc{cost}(\alpha) \leq \textsc{Opt}(\textsc{Dual}) = \textsc{OPT}(\textsc{Primal}) \leq \textsc{cost}(C^*)$. Combined with the fact that $\textsc{cost}(\alpha) = \textsc{cost}(C^*)$, this will show that clustering is an optimal solution to the DP-Facility problem.

Consider the linear programming relaxation to DP-Facility problem. This LP is an adaption of the classical LP used for the facility location problem considered in [271][Ch. 17].

$$\min \sum_{i \in F} \sum_{j \in C} e(i,j) \cdot z_{i,j} + \lambda \sum_{i \in F} y_i$$

$$\sum_{i \in F} z_{ij} \geq 1 \qquad \text{for all } j \in C$$

$$y_i - z_{ij} \geq 0 \qquad \text{for all } j \in C$$

$$z, y \geq 0$$

The above program contains two variables z_{ij} indicating if client j is connected to facility i and variables y_i indicates if facility i is open. In particular, every feasible solution to the DP-Facility problem is a candidate solution to the above LP. The dual to the above program is given below:

$$\begin{aligned}
&\max \sum_{j \in C} \alpha_j \\
&\sum_{j} \beta_{ij} \leq \lambda \qquad \text{for all } i \in F \\
&(\alpha_j - e(i,j)) \leq \beta_{ij} \qquad \text{for all } i \in F, j \in C \\
&\alpha, \beta \geq 0
\end{aligned}$$

For each cluster C_i, and each point in the cluster $x \in C_i$, $\alpha_x = ((\delta - 2)R + e(c_i^*, x)/r$ where r is the size of cluster $r := |C_j|$. By δ-separability assumption, we can deduce that $\beta_{\tilde{i}x} = 0$ for all other clusters $C_{\tilde{i}}^* \neq C_i^*$. However, for all $x \in C_i$, we will have $\sum_{x \in C_i} \beta_{ix} = r \cdot \frac{\lambda}{r} = \lambda$. This shows that α is a valid dual to the LP. ■

4.7.4 Proof of Proposition 2

Proposition 2 *Let $f : \mathcal{P}(\boldsymbol{X}) \times \mathcal{P}(\boldsymbol{X}) \to \mathbb{R}$ be a linkage function that is symmetric and injective, $C_1, C_2, C_3, C_4 \subset \boldsymbol{X}$, $f(C_1, C_2) = f(C_3, C_4) \iff (C_1 = C_3 \wedge C_2 = C_4) \vee (C_1 = C_4 \wedge C_2 = C_3)$ (i.e., the linkage between each pair of nodes is unique). Let $\mathcal{T}$ be the tree formed by Algorithm 1 and let f also satisfy reducibility [55], $\forall C, C', C'' \in \mathcal{T}$, $f(C, C') \geq \max\{f(C, C''), f(C', C'')\} \implies \max\{f(C, C''), f(C', C'')\} \geq f(C \cup C', C'')$ then there exists a sequence of threshold $t_1, \ldots, t_r$ such that the tree formed by $\bigcup \text{SCC}(\boldsymbol{X}, f, \{\tau_1, \ldots, \tau_r\})$ is the same as $\mathcal{T}$.*

Proof. Each node in the tree, $\mathcal{T}$, produced by HAC has an associated linkage function score. For node $C \in \mathcal{T}$, we abuse notation and call this $f(C)$. We define the threshold-based rounds for SCC such that $t_1, \ldots, t_r$ to be the values, $\{f(C) + \epsilon | C \in \mathcal{T}\}$ sorted in ascending

order. Because f is reducible and injective, there is a unique pair of nodes that will be merged in each round. This pair will correspond exactly to the pair that is merged by HAC in the corresponding round. It follows that the resulting tree structures will be identical.

We can analyze HAC using the same round notation used in SCC. HAC proceeds by finding the closest two clusters according to g in the previous round's clustering $\mathcal{C}^{(i-1)}$, and joining them to form a new clustering for the next round $\mathcal{C}^{(i+1)} := \{\mathcal{C}^{(i)} \setminus \{C_a, C_b\}\} \cup (C_a \cup C_b)$ $s.t.$ $C_a, C_b = \text{argmax}_{C_1, C_2 \in \mathcal{C}^{(i)}} d(C_1, C_2)$. The final hierarchical clustering is given as $\{\cup\, \mathcal{C}^{(i)}\}_{i=1}^{N-1}$. The round-based threshold is set to the linkage cost in the corresponding round of HAC to ensure that only a single merger is made by SCC since f is injective. ■

CHAPTER 5

DAG-STRUCTURED CLUSTERING

A key limitation to the hierarchical clustering approaches considered up until this point is their inability to represent non-nested overlapping clusters. For example, Figure 5.1 shows a subset of the DAG structure built by our proposed method when clustering word embeddings. The structure can simultaneously represent two senses of the word shepherd (the type of dog and the profession). Such a structure cannot be represented by a tree. In this section, we present algorithms for building DAG-structured clusterings at scale using a simple extension of the reciprocal nearest neighbors algorithm [217].

5.1 Building DAG-Structured Clustering

Akin to the reciprocal nearest neighbor algorithm [217], we present a simple round-based algorithm for building DAG-structured clusterings. First, we will depart from the hierarchical clustering tradition by removing the assumption that each round of the algorithm will produce a flat clustering. Instead, each round will produce a *cover* of the dataset. Points may be assigned to multiple clusters[1] in the given round. We use $\mathcal{S}^{(i)}$ to refer to the cover produced in round i.

The algorithm begins with each data point in its own cluster. In round i, each cluster C finds its nearest neighbor C' among the members of the cover $\mathcal{S}^{(i-1)}$. These two are merged

[1]We use cluster to refer to member sets of a cover.

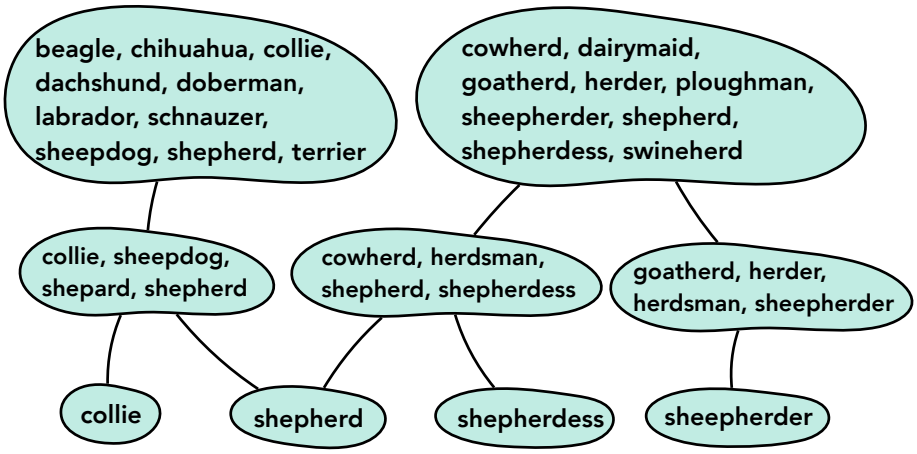


Figure 5.1: **DAG-structured Clustering.** A substructure of the clustering produced by our proposed algorithm on a dataset of word vectors. Observe how the word shepherd appears in both the cluster of dog breeds as well as the cluster of farm professions.

to form a new node $C \cup C'$ in the following round. This is done for all pairs of nearest neighbors:

$$\mathcal{N}^{(i)} = \{(C, C') \mid C' = \underset{B \in \mathcal{S}^{(i-1)}}{\operatorname{argmax}} f(C, B)\} \tag{5.1}$$

$$\mathcal{S}^{(i)} = \{C \cup C' | (C, C') \in \mathcal{N}^{(i)}\}. \tag{5.2}$$

Observe that non-reciprocal nearest neighbor relationships are the source of a cluster having more than one parent in the given round. The cluster of points, C, may be the nearest neighbor of many other clusters C'. Any cluster $C \in \mathcal{S}^{(i-1)}$ may have multiple supersets in $\mathcal{S}^{(i)}$, and that every nearest neighbor cluster C' of $C \in \mathcal{S}^{(i-1)}$ will lead to a unique superset of $C \in \mathcal{S}^{(i)}$. From the data structure point of view, the node corresponding to cluster C will have a parent corresponding to the cluster of $C \cup C'$ for each unique nearest neighbor cluster C'.

We refer to this algorithm as **L**attices by **L**everaging **A**gglomerations and **M**ultiple **A**ncestors (LLAMA) because of its ability to build clustering structures where points have multiple ancestries. We take the number of rounds used in the algorithm as an optional hyperparamater. Pseudocode is given in Algorithm 11.

The algorithm can be implemented to utilize parallelism. The computation of $\mathcal{N}^{(i)}$ is trivially parallelizable. The computation of $\mathcal{S}^{(i)}$ can also be parallelized using $\mathcal{S}^{(i-1)}$ and $\mathcal{N}^{(i)}$.

5.2 Limiting the Size of the DAG-structures

The size of the DAG-structures discovered by LLAMA will depend on the number of parents of each node. The number of parents can be be bounded by an additional hyperparameter in the algorithm, p. We can then adapt the LLAMA algorithm to select at most p parents for each node. We propose to do this by finding the top p candidate parents in $\mathcal{N}^{(i)}$ for each particular cluster C, according to $f(\cdot,\cdot)$, we refer to this set as P_C. We will then restrict the entries in $\mathcal{N}^{(i)}$ to be those tuples (C, C') such that $(C, C') \in \mathsf{P}_C$ and $(C, C') \in \mathsf{P}_{C'}$. In words, this means the candidate parent is in the top p candidate parents for both C and C' lists. Let $\mathcal{N}_C^{(i)}$ be the entries in $\mathcal{N}^{(i)}$ that contain C, then in our set-based notation, this is:

$$\mathsf{P}_C = \underset{(B,B')\in\mathcal{N}_C^{(i)}}{\text{argtopk}} \; f(B, B') \tag{5.3}$$

$$\mathcal{N}^{(i)\prime} \leftarrow \{(C, C') \mid (C, C') \in \mathsf{P}_C \cap \mathsf{P}_{C'}\} \tag{5.4}$$

We then update $\mathcal{S}^{(i)}$ to include both the new sets from $\mathcal{N}^{(i)\prime}$ as well as singleton clusters that were assigned no parent:

$$\mathcal{S}^{(i)\prime} = \{C \cup C' \mid (C, C') \in \mathcal{N}^{(i)\prime}\} \cup \{C | \mathsf{P}_C \cap \mathcal{N}^{(i)\prime} = \emptyset\} \tag{5.5}$$

Algorithm 11 LLAMA

Input: $\boldsymbol{X}$: dataset, f: set similarity function, L: number of rounds (optional, default ∞)
Output: $\mathcal{D}$: a DAG-structured clustering
$\mathcal{S}_0 \leftarrow \{\{x\} \mid x \in \boldsymbol{X}\}$
$\mathcal{D} \leftarrow \mathcal{S}_0$
for i **from** 1 **to** L **do**
    $\mathcal{N}_i \leftarrow \{(C, C') \mid C' = \text{argmax}_{C'' \in \mathcal{S}_{i-1}} f(C, C'')\}$
    $\mathcal{S}_i \leftarrow \{C \cup C' \mid (C, C') \in \mathcal{N}_i\}$.
    $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{S}_i$
    if $|\mathcal{S}_i| = 1$; **return** $\mathcal{D}$
return $\mathcal{D}$

5.3 Algorithmic Details

To make the construction of $\mathcal{N}^{(i)}$ more efficient, we build k-nearest neighbor graphs with respect to the *similarity* function (`sim`) for a dataset. We weight the edges of the graph with the similarity between the points. Edges that are missing from the graph are assumed to have 0 similarity. We can use this k-nearest neighbor graph with vertices $\boldsymbol{X}$ and edges E to define an analogous average linkage: $\frac{1}{|A||B|} \sum_{a \in A} \sum_{b \in B} w_{ab} \mathbb{I}[(a, b) \in E]$ where $\mathbb{I}[(a, b) \in E]$ determines if the edge is in the graph. When computing $f(\cdot, \cdot)$ we can then restrict our consideration for candidate nearest neighbors in $\mathcal{N}^{(i)}$ to connected nodes. When considering candidate nearest neighbors of a cluster, we consider other clusters such that there is at least one edge in the nearest neighbor graph between the points in the clusters.

Previous work [216, 217] has shown that we can efficiently update the linkage function values between newly merged clusters using the linkage function values of existing ones. For tree structures, these are well known [216] and provide for massive speedups in the methods as the average linkage can be computed as the sum of values for cluster pairs (instead of having to consider all of their descendant points). However, if the clusters are overlapping (as in our setting) the standard update rules for certain linkages will no longer be technically correct such as for average linkage. Empirically, however we find that approximating the average linkage by using the standard update rules from [216] achieves good performance. Interestingly, for average linkage, this approximation looks very much like doing a bag-

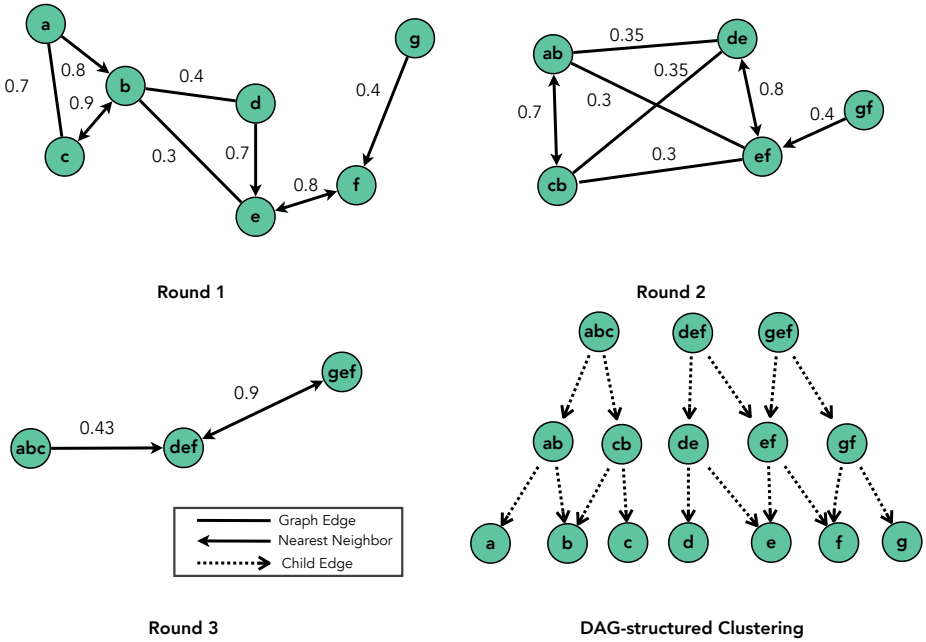


Figure 5.2: **LLAMA Algorithm.** An example of our proposed algorithm applied to a toy example graph. The nearest neighbor relationships of each round are shown. The resulting structure is shown on the far right.

based average linkage where the number of times edges are double counted is a function of number of times the nodes has appeared in the overlap of two nodes. Table 5.2 provides a comparison between using this approximate computation using the update rule and the exact version.

5.4 Theoretical Analysis

In this section, we show that LLAMA is able to accurately recover the target partition when data follows model-based separation (Assumption 1). We then demonstrate that LLAMA is able to recover the target partition for noisy model-based separation (Assumption 3) while tree-based methods such as Grinch and HAC cannot.

To show that LLAMA can recover DAG structures that contain the target model-based separated partition $\mathcal{C}^\star$, we first make the following observation about the pairs of nearest neighbors that are merged in each round:

Lemma 5. *Given a dataset $\boldsymbol{X}$ and a symmetric linkage function f such that $\boldsymbol{X}$ is model-based separated with respect to f, let $\mathcal{C}^\star$ be the target partition corresponding to the separated data. In each round of* LLAMA*, each pair of nearest neighbors $(C, C') \in \mathcal{N}^{(i)}$, will satisfy:*

$$\exists C^* \in \mathcal{C}^\star \text{ s.t. } C \cup C' \subseteq C^*, C^* \subseteq C, \text{ or } C^* \subseteq C'.$$

Please see Section 5.6.1 for the proof.

Now, we show that our proposed approach can cluster the same class of data represented by trees (model-based separation):

Theorem 8. *Given a dataset $\boldsymbol{X}$ and a linkage function f such that $\boldsymbol{X}$ is model-based separated with respect to f, let $\mathcal{C}^\star$ be the target partition corresponding to the separated data. Let $\mathcal{D}$ be the DAG-structured clustering produced by* LLAMA *(Alg. 11), then $\mathcal{C}^\star$ is a $\mathcal{D}$ consistent partition, $\mathcal{C}^\star \subset \mathcal{D}$.*

Please see Section 5.6.2 for the proof.

We have now seen that LLAMA is at least as expressive as tree-based methods for clustering model-based separated data. Now we turn to a data separation assumption that is not recovered by tree-based methods. While model-based separation is flexible and relatively loose compared to strict separation, an aspect of it that is overly rigid is its assumption that *every* point in a cluster has some point in their cluster that is closer than every point outside of their cluster. We now want to consider a loosening of this restriction to allow *some* points to have nearest neighbors outside their clusters, *noisy model-based separation* (Assumption 3).

We now prove that LLAMA recovers a DAG-structure with the noisy model-based separated partition.

Proposition 3. *Given a dataset $\boldsymbol{X}$ and a linkage function f such that $\boldsymbol{X}$ is model-based separated with respect to f, let $\mathcal{C}^\star$ be the target partition corresponding to the noisy*

model-based separated data. Let $\mathcal{D}$ be the DAG-structured clustering produced by LLAMA *(Alg. 11), then $\mathcal{C}^\star$ is a $\mathcal{D}$ consistent partition, $\mathcal{C}^\star \subset \mathcal{D}$.*

Please see Section 5.6.3 for proof.

Finally, we can see that tree-based methods such as Grinch and HAC cannot recover noisy model-based separated data:

Proposition 4. *There exists a datasets $\boldsymbol{X}$ and symmetric linkage function f such that $\boldsymbol{X}$ is noisy model-based separated wrt f, let $\mathcal{C}^\star$ be the target partition corresponding to the noisy model-based separated data. HAC and* GRINCH *produces a structure $\mathcal{T}$ such that $\mathcal{C}^\star$ is not a tree consistent partition, $\mathcal{C}^\star \not\subset \mathcal{T}$.*

Please see Section 5.6.4 for proof.

5.4.1 Complexity

We analyze the space and time complexity of LLAMA. See Section 5.6.5 for proofs for each statement.

Proposition 5. (Space Complexity). *Given a dataset of N points,* LLAMA *produces DAG-structured clusterings with at most $O(N^2)$ nodes.*

Proposition 6. (Time Complexity). *Given a dataset of N points, R rounds of* LLAMA *requires at most $O(R * N^2)$ linkage function computations.*

Proposition 7. (Number of Rounds). *Let $\mathcal{C}^\star$ be the target partition of a dataset that is (noisy) model-based separated, let K be the size of the largest cluster in $\mathcal{C}^\star$. $K = \max_{C \in \mathcal{C}^\star} |C|$. After K rounds,* LLAMA *produces a structure that contains $\mathcal{C}^\star$.*

We note that while the worst-case complexity is quadratic in the number of datapoints, we find that in practice the structures are much smaller than this and typically require a much smaller (usually constant, e.g. 50 to 200) number of rounds to get meaningful structures.

		LLAMA		RcNN		Affinity		Grinch
		Sing.	Avg.	Sing.	Avg.	Sing.	Avg.	
Jacc/node	**ALOI**	0.067	**0.117**	0.068	0.037	0.027	0.027	0.052
	ILSVRC (Sm.)	0.154	**0.284**	0.146	0.076	0.044	0.047	0.171
	Speaker	0.271	**0.329**	0.231	0.227	0.175	0.177	0.257
	ImageNet	0.154	0.154	**0.178**	0.173	0.171	0.169	0.165
	ILSVRC (Lg.)	0.023	**0.023**	-	0.005	0.002	0.003	-
Jacc/pt	**ALOI**	**0.700**	0.560	0.594	0.593	0.648	0.518	0.509
	ILSVRC (Sm.)	0.559	**0.655**	0.393	0.626	0.537	0.555	0.575
	Speaker	0.485	**0.582**	0.467	0.563	0.430	0.447	0.564
	ImageNet	0.219	**0.219**	0.201	0.218	0.199	0.199	0.208
	ILSVRC (Lg.)	0.540	0.604	-	**0.621**	0.546	0.530	-
Jacc/lbl	**ALOI**	**0.759**	0.647	0.704	0.669	0.713	0.605	0.615
	ILSVRC (Sm.)	0.638	**0.728**	0.528	0.707	0.617	0.638	0.661
	Speaker	0.665	**0.726**	0.659	0.713	0.607	0.615	0.711
	ImageNet	0.390	**0.399**	0.366	0.384	0.360	0.360	0.372
	ILSVRC (Lg.)	0.623	0.677	-	**0.702**	0.625	0.615	-

Table 5.1: **Clustering Benchmarks**. *Precision metric* is Jacc/node and *Recall metrics* are Jacc/pt and Jacc/lbl.

5.5 Experiments

We compare the performance of LLAMA to state-of-the-art methods for hierarchical and DAG-structured clusterings. We evaluate the effectiveness of each at recovering ground-truth labeled data. We further attempt to automatically reconstruct the DAG-structure WordNet [206] from vector representations of words using LLAMA.

5.5.1 Clustering Benchmarks

First, we consider datasets where each point is assigned one ground truth cluster. In this experiment, we hope to understand if the clusters for each point that are discovered by LLAMA are better aligned with the underlying data than those of competing methods.

Following previous work [170], we run experiments on publicly available large scale hierarchical clustering benchmark datasets. We evaluate on the following datasets: **Speaker**, feature vectors representing audio signals of spoken voices from different speakers (each speaker is a ground truth cluster) [132]; **ALOI** (Amsterdam Library of Object Images), histogram features of toy objects [123]; **ILSVRC (Sm.) (50K subset)** and **ILSVRC (Lg.)**

	Avg. link.	LLAMA Approx.	LLAMA Exact.	RcNN	OHC
Jacc/node	**ALOI**	**0.152**	0.100	0.044	0.091
	ILSVRC	**0.346**	0.2647	0.090	0.197
	Speaker	0.405	**0.408**	0.271	0.349
	ImageNet	0.275	**0.324**	0.280	0.268
Jacc/pt	**ALOI**	0.984	0.979	0.950	**0.990**
	ILSVRC	0.975	0.973	**0.976**	0.924
	Speaker	0.804	**0.832**	0.813	0.827
	ImageNet	0.593	**0.604**	0.583	0.567
Jacc/lbl	**ALOI**	0.897	0.892	0.880	**0.908**
	ILSVRC	**0.936**	0.935	**0.936**	0.904
	Speaker	0.844	**0.860**	0.853	0.843
	ImageNet	0.688	**0.698**	0.690	0.690

Table 5.2: **Comparison to OHC**. We sample datasets of 1000 points and report results with average linkage. Aff. and Grinch are outperformed by other methods. We compare the two variants of average linkage (Section 5.3).

(1.2M Images) Inception embeddings from the ImageNet ILSVRC 2012 dataset [242]; **Imagenet** a sample of 100k images from all 17K classes present in ImageNet. Table 5.6 provides the statistics for each dataset used in the clustering and cover-based evaluations.

We evaluate against tree-based following methods: **Affinity clustering (Aff.)** [31], a round-based bottom-up hierarchical clustering method that connects each point to its nearest neighbor in a single round and builds nodes in a tree based on connected components in this 1-nearest neighbor graph; **Grinch** [210], an online hierarchical clustering method that performs tree re-arrangements after each point is inserted; **Reciprocal Nearest Neighbors (RcNN)** [217] (Section 2.6.2).

Each dataset uses cosine similarity. LLAMA, RcNN, and Affinity all make use of linkage functions that use k-nearest neighbor graph sparsification. This technique precomputes a k-NN graph over the dataset so as to make the argmax operations in the algorithm more efficient (Section 5.3). For RcNN, and Aff. we report results with both single and average linkage. For LLAMA, we use single and a approximation of average linkage that supports a

		LLAMA	RcNN	Aff.	Grinch
Jacc/node	**EURLex-4k**	**0.182**	0.156	0.142	0.111
	Bibtex	**0.081**	0.025	0.017	0.024
	Wiki10-31K	0.298	0.404	**0.411**	–
	Delicious	**0.068**	0.027	0.020	0.026
	MediaMill	**0.009**	0.004	0.003	–
Jacc/pt	**EURLex-4k**	0.172	**0.180**	0.143	0.061
	Bibtex	0.178	**0.198**	0.166	0.067
	Wiki10-31K	**0.184**	0.104	0.168	–
	Delicious	0.108	**0.129**	0.124	0.109
	MediaMill	0.335	**0.342**	0.339	–
Jacc/lbl	**EURLex-4k**	**0.466**	0.455	0.424	0.332
	Bibtex	0.172	**0.179**	0.138	0.089
	Wiki10-31K	**0.415**	0.392	0.361	–
	Delicious	0.090	**0.091**	0.076	0.056
	MediaMill	**0.099**	0.096	0.089	–

Table 5.3: **Covering Benchmarks**. The datasets for which Grinch did not finish are marked with dashes. All methods use average linkage.

more efficient implementation (Section5.3). For Grinch, which does not use k-NN graph sparsification, we use its most efficient (and best performing) implementation that uses a centroid-based linkage.

Table 5.1 shows the results for this experiment. We observe that LLAMA outperforms the other methods in all but three of the dataset/metric combinations. We hypothesize that the improvements observed by LLAMA are due to the DAG structure's flexibility in representing alternative clusterings. Importantly, LLAMA performs better on both the precision-based (Jacc/node) and recall-based (Jacc/pt, Jacc/lbl) metrics. This indicates that the structures discovered by the method include, on average, nodes that are better aligned with the ground truth clustering (recall) and fewer spurious nodes that do not have significance with respect to the underlying data (precision). The dashed cells indicate the algorithm exceeded our 10hr, 150GB RAM limit.

Leaf Node	Ancestors Discovered by LLAMA
blossoming	{blossom, blossoming}, {budding, blossoming}, {budding, emergent, emerging, fledgling, incipient, nascent, blossoming}, {abloom, blooming, flowered, flowering, bloom, bloomer, bloomers, blossom, blossoming}, {abloom, autumn-flowering, blooming, early-blooming, early-flowering, fall-blooming, flowered, flowering, half-hardy, late-blooming, late-flowering, planted, seeded, sown, spring-blooming, sprouted, summer-bloom}
disloyal	{disloyal, allegiance, disloyalty, loyalty}, {anti-american, disloyal, pro-american, seditious, traitorous, treasonable, treasonous, un-american, unpatriotic, collaborationist, disloyalty, incitement, quisling, sedition, traitor, treason, treasonist, turncoat }, {adulterous, disloyal, faithless, unfaithful, adulterer, adultery, allegiance, commitment, dedication, devotedness, devotion, disloyalty, faithfulness, faithlessness, fealty, fidelity, fornication, infidelity, loyalty, unfaithful}

Table 5.4: **Example Clusters Discovered by LLAMA**. Sample nodes from the DAG-structured clustering. We observe that the algorithm discovers interesting overlapping components clusters in the vector space with different lineages of leaf nodes revealing multiple senses of each word.

To compare with bottom-up DAG-structured clustering algorithms that operate in a sequential fashion, we compare to **Overlapping Hierarchical Clustering (OHC)** [156]. OHC is a DAG-structured clustering method that considers agglomerations, like HAC, one edge at a time and uses a distance threshold to determine whether a node should participate in multiple agglomerations. We could not get results for OHC on the above datasets in the 10 hours/dataset we allot to each method as these are much larger than the ones used by in the original paper. To provide a comparison to OHC, we compare the methods on a random subset of 1000 points and evaluate the methods on these subsets. We run a hyperparameter sweep over the parameters of OHC (merging criterion and batch size) and report the best performing OHC result for each dataset in Table 5.2.

In the experiments, we use 50 rounds for LLAMA and restrict the number of parents to be 5. RcNN needs around 100 rounds for convergence on all except ILSVRC (Lg.) needing 200 rounds. In Section 5.5.6, we analyze hyperparameters of LLAMA including the number of neighbors in the k-NN sparsification and the number of rounds used.

5.5.2 Covering Benchmarks

Next, we take extreme multi-label classification benchmark datasets for which the ground truth is a *cover* rather than a partition (See Table 5.6): **MediaMill**, **Delicious**, **BiBTeX**, **EURLex-4k**, **Wiki10-31K**.

We compare to the same set of algorithms as used in Section 5.5.1. We use the same Jaccard-based metrics as before since these metrics can be applied to both partition and cover-based labelings of data. We use the same experimental settings that are used for the partition-based benchmarks. Table 5.3 provides the results for this experiment. We observe that our proposed method either outperforms or is competitive with tree-based metrics on all datasets/metrics.

5.5.3 WordNet Reconstruction

We perform analysis on the task of automatically building lexical resources. WordNet [113, 206] is a manually curated resource that records, among other information, *synsets*, sets of English words that are synonymous. Words may be polysemous and so the same word spelling may exist in two synsets. We attempt to recover these synsets from the vector data using LLAMA and other approaches.

We select the subset of WordNet for which the word type has a representation in the fasttext model (leaving 64K words) [205]. We again use average linkage with cosine similarities as in the prior experiments. Each word has a single embedding for its spelling.

	LLAMA	**LLAMA**	**RcNN**	**Aff.**	**Grinch**
Rounds	50	5			
Jacc/node	0.307	0.474	0.571	**0.667**	0.532
Jacc/pt	**0.714**	**0.714**	0.695	0.645	0.664
Jacc/lbl	**0.869**	**0.869**	0.857	0.823	0.839

Table 5.5: **WordNet** Reconstruction evaluation metrics.

This means that both senses of the word *crane* are represented by the same point. We take the synset labels of these words as the ground truth labels.

Table 5.5 provides the quantitative results. We show results for two variants of LLAMA, one with 50 rounds and another with 5 rounds. We hypothesized that the structure of the synsets is relatively fine grained and so introducing additional rounds of the algorithm that adds larger nodes is the reason for the decrease in the precision-based mean Jaccard per node metric.

Despite each word having a single point representation, we are able to discover alternative senses of various words using LLAMA (Figure 5.1 and Table 5.4). We showed an example (Figure 5.1) of how the word shepherd appears in both a cluster of dog breeds as well as in a cluster of farming professions. In Table 5.4, we provide additional examples where disloyalty is described to cluster with words meaning tyranny as well as adultery and blossoming is clustered with the floral sense and the emergent sense. These results highlight how in high dimensional spaces these word vectors are able to be close to each other via many different directions. Further, they indicate examples where such overlapping cluster assignments would not be possible in tree-based methods.

5.5.4 Discovering Topics In US Patents

We also apply LLAMA to US Patent data. We represent each patent by a Sent2Vec [226] embedding of the title of the patent. We then run LLAMA with average linkage on these patent titles. Interesting, we find that LLAMA can discover meaningful overlapping clusters of patent topics. We show two examples in Figure 5.3. Interesting, we find that patents related to *Headphones* appear in both clusters related to *Hearing Aids* as well as *Acoustic Systems*. However, the *Hearing Aids* and *Acoustic Systems* clusters are otherwise non-overlapping. We observe a similar scenario in the clusters related to *Locks*. We find three clusters about *Safes and locks*, *Lock extractors*, and *Luggage / portable locks* that are disjoint other than their overlap on general *lock* related patents. The Sent2Vec embeddings provide

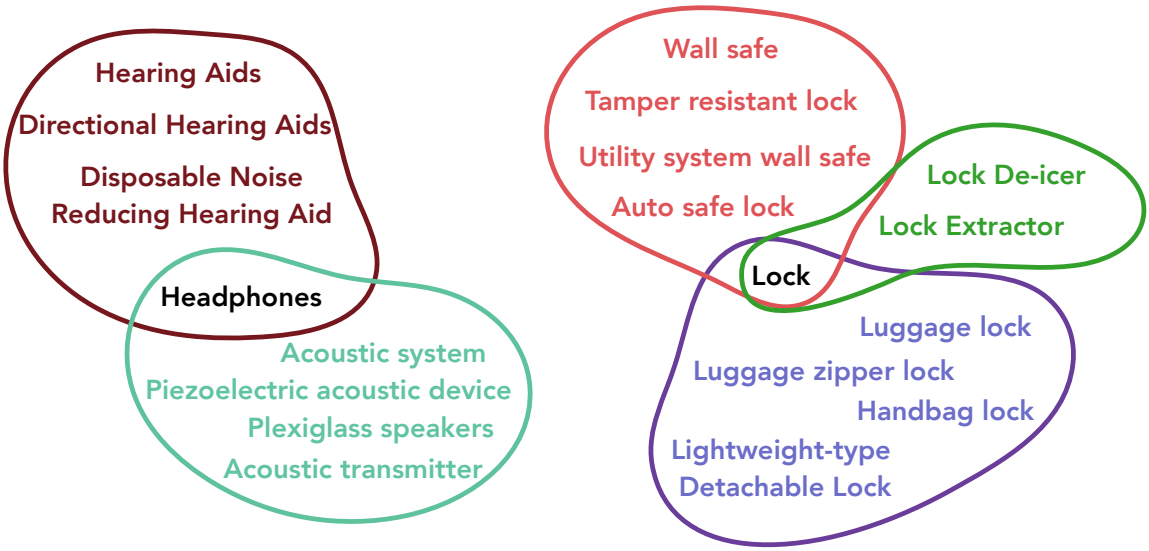


Figure 5.3: **LLAMA discovering topics in Patent Data**. Two sample overlapping clusters discovered when LLAMA is applied to 500K US Patent Titles, which are represented by Sent2Vec embeddings [226].

a single representation per patent, yet LLAMA is able to discover meaningful overlapping clusters, highlighting the complexity of the high dimensional vector space of the titles.

5.5.5 Analysis of Jaccard-based Clustering Metrics

Dendrogram Purity [146] (Eq. 2.4) is a metric that is often used to evaluate the quality of a hierarchical clustering of a dataset which has a ground truth flat partition. Rather than demanding a particular flat clustering be extracted from the tree structure, dendrogram purity evaluates the quality of the tree consistent partitions encoded in the hierarchical clustering.

We note that there are trivial DAG structures which would achieve perfect dendrogram purity. In particular, the DAG structure which contains the cluster for each pair of points in the dataset.

We are interested to understand which of the Jacc/pt, Jacc/lbl, Jacc/node is most correlated to dendrogram purity for trees. To analyze this, we sample synthetic data from Dirichlet Process Mixture Models (DPMMs) with spherical variance. We sample 10 datasets from 75 different DPMM hyperparameter settings in $\mathbb{R}^{10}$ for a total of 750 datasets. The

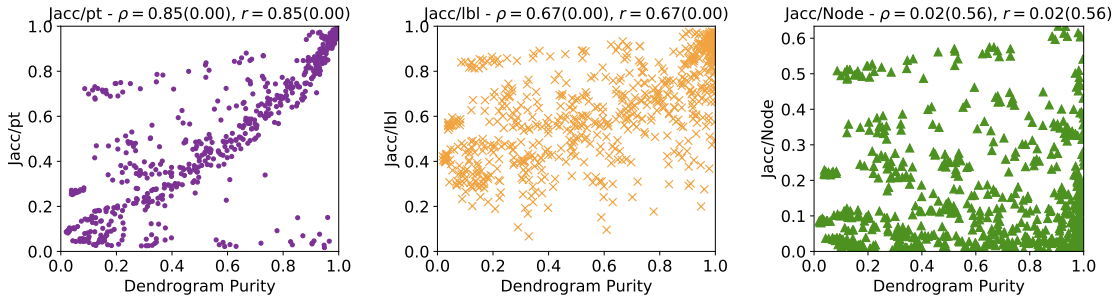


Figure 5.4: **Dendrogram Purity and Jaccard Metrics on Synthethic Data**. We report the Spearman (ρ) and Pearson r correlation for each and p value in parenthesis. We observe that Jacc/pt is well correlated with dendrogram purity. While the other metrics are not correlated, this does not diminish our interest in them as metrics. Jacc/node captures how precise or compact the structures are, unlike dendrogram purity. Jacc/lbl measures at the label level how well represented the ground truth clusters are. Unlike Jacc/pt and dendrogram purity, Jacc/lbl weights each ground truth cluster equally independent of the size of the cluster. As the data here has CRP distributed cluster sizes, it is no surprise that Jacc/lbl looks quite different than Jacc/pt.

75 settings come from the cartersian product of (number of points ({100, 1000, 5000}), variances ({0.25, 0.4, 0.5, 0.75, 1.0}), and alpha parameters of CRP ({1, 5, 10, 25, 100}). For each dataset we run the best tree-based method, reciprocal nearest neighbors and report all metrics. We plot each metric against dendrogram purity in Figure 5.4 and observe that Jacc/pt is most correlated to dendrogram purity. We note that dendrogram purity is by no means the only metric that we are interested in and so the lack of correlation for the other two metrics is not a negative result, it simply implies that they capture something different about the structures. Furthermore, for this particular choice of generative models, which encourages rich-get-richer cluster sizes, it is no surprise that Jacc/lbl looks quite different than Jacc/pt. Similarly, Jacc/node measures the compactness of the structure and so captures properties that dendrogram purity does not.

We also show these same plots for the results of all three tree-based methods compared (reciprocal nearest neighbor, Affinity, Grinch) on the real world clustering datasets. The results follow a similar trend as the synthetic data and are shown in Figure 5.6.

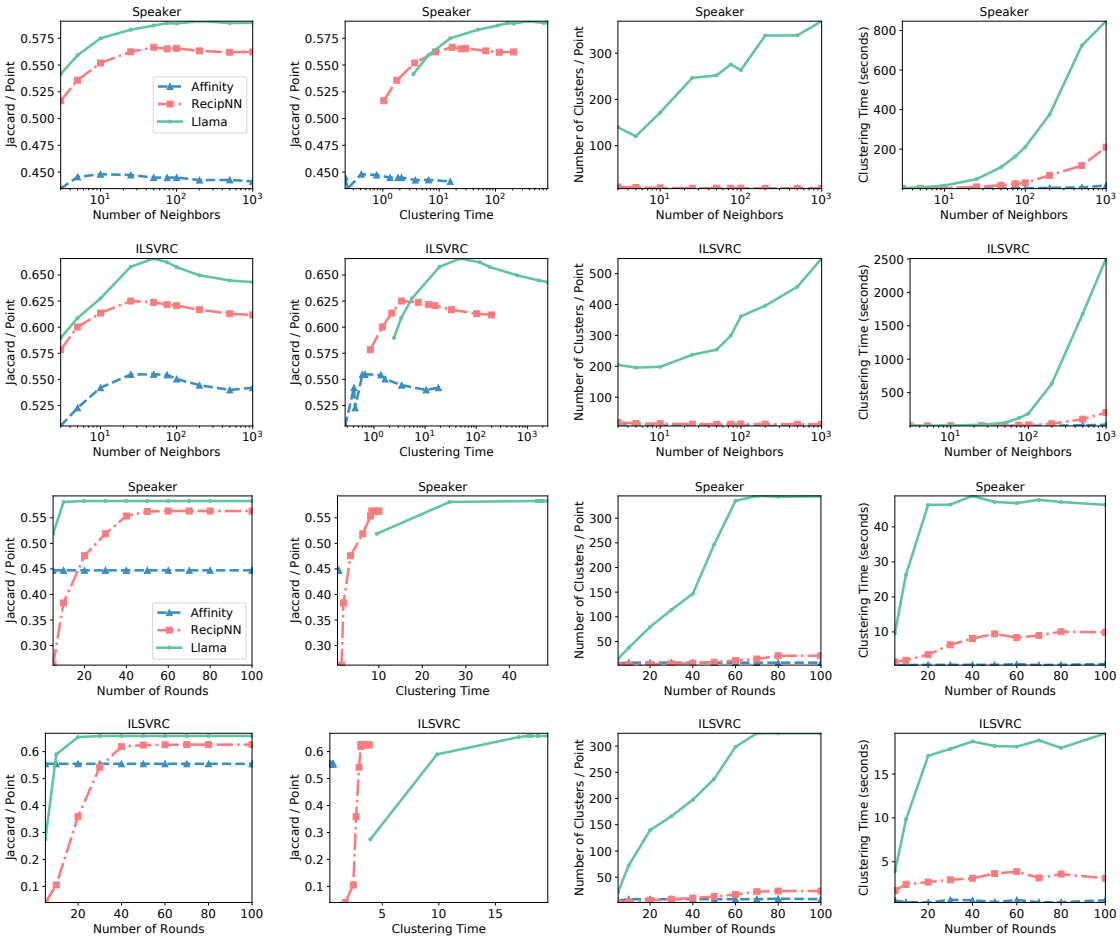


Figure 5.5: **Hyperparameter Analysis**. We compare performance on the Speaker and ILSVRC (Sm.) datasets using various numbers of rounds and various settings of the number of nearest neighbors in the nearest neighbor graph. We observe comparable performance across various kinds of nearest neighbors. We observe that around 20-40 rounds is required for competitive performance of the metrics. Importantly, while the complexity of LLAMA does grow faster than the other methods in terms of time and number of nodes, we observe good performance can be achieved in the parts of the time/space curves that are much closer to tree-based methods.

		Num. Points	Num. Labels	Dim
Partition-based	**ALOI**	108K	1000	128
	ILSVRC (Sm.)	50K	1000	2048
	Speaker	36.5K	4958	6388
	ImageNet	100K	17K	2048
Cover-based	**EURLex-4k**	19K	3993	5000
	Bibtex	7K	159	1836
	Delicious	16K	983	500
	MediaMill	43.9K	101	120

Table 5.6: **Dataset Statistics**. The sizes and number of labels for the datasets used in DAG experiments.

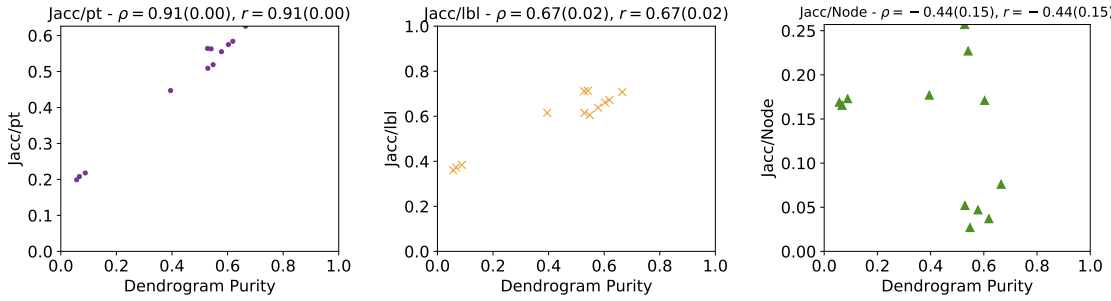


Figure 5.6: **Dendrogram Purity and Jaccard Metrics on Real Data**. As in Figure 5.4, we report the values of the metrics in this case on the hierarchical clustering benchmark datasets.

5.5.6 Hyperparameter Analysis

We analyze two hyperpameters of LLAMA, RecipNN, and Affinity, the number of neighbors of the nearest neighbor graph (described above) and the number of rounds of the algorithm used. Figure 5.5 shows the results. We observe that around 20-40 rounds are required for competitive performance. We observe that the number of nearest neighbors between 3 and 1000 does not lead to major variation in performance. We observe that while LLAMA can become much more expensive than tree structures when the number of rounds or graph density becomes very large, the algorithm does not much more time than the tree-based methods to achieve better-than-tree structure performance.

5.5.7 Running Time Analysis

In Table 5.7, we report a timing comparison of running 100 rounds of LLAMA and Reciprocal NN algorithm on the clustering benchmarks. We use 10 threads in parallelizing each algorithm's computation of the neighbors and linkage function values. For LLAMA, we use the comparably efficient approximate average linkage. We report the time of clustering the pre-computed sparse graph (which is done using ScaNN [137]).

	Running Time (s)		**Avg. Clusters / Point**	
	RcNN	**LLAMA**	**RcNN**	**LLAMA**
ALOI	12.32	7.58	18.474	127.621
Speaker	10.43	53.63	19.819	238.21
ILSVRC (Sm.)	3.955	10.66	24.18	162.35
ImageNet	16.754	224.533	21.513	600.67
ILSVRC (Lg.)	86.32	495.19	44.387	356.311

Table 5.7: **Running Times & Structure Size**. The running time of the two algorithms on each of the clustering benchmarks. Interestingly, LLAMA takes less time on the ILSVRC (Sm.) dataset than the Speaker dataset, despite it being larger. We hypothesize that the time taken by LLAMA is directly impacted by the underlying structure of the dataset's similarity graph and with more separation in the data (as seems to be the case here), LLAMA can be more efficient. For the same runs as the timing numbers, we report the number of average number of clusters each point has been assigned to in the structures, which share a similar trend.

5.6 Proofs and Additional Details

5.6.1 Proof of Lemma 5

Lemma 5 *Given a dataset $\boldsymbol{X}$ and a linkage function f such that $\boldsymbol{X}$ is model-based separated with respect to f, let $\mathcal{C}^\star$ be the target partition corresponding to the separated data. In each round of* LLAMA*, each pair of nearest neighbors $(C, C') \in \mathcal{N}^{(i)}$, will satisfy either:*

1. *$\exists\, C^\star \in \mathcal{C}^\star$ such that $C \subseteq C^\star$ and $C' \subseteq C^\star$, or*
2. *$\exists\, C^\star \in \mathcal{C}^\star$ such that $C^\star \subseteq C$ or $C^\star \subseteq C'$.*

Proof. We will prove this by induction. The first round of the algorithm, in which each point sits in its own cluster, satisfies the above property. Now let us assume that $\mathcal{N}^{(i-1)}$ has the above property. We want to show that $\mathcal{N}^{(i)}$ has the property as well. Each $C \in \mathcal{S}^{(i-1)}$ finds its nearest neighbor in $\mathcal{S}^{(i-1)}$ according to the linkage function f, we denote this as $C' = \text{argmin}_{C'' \in \mathcal{S}^{(i-1)} \setminus C} f(C, C'')$. There are three cases.

Case A: $\exists\, C^\star \in \mathcal{C}^\star$**, s.t.,** $C = C^\star$. In this case, the node C corresponds exactly to the ground truth cluster. For any node that it pairs with it will satisfy Condition (2) above.

Case B: $\exists\, C^\star \in \mathcal{C}^\star$**, s.t.,** $C^\star \subseteq C$. In this case, as a ground truth cluster is already consumed by the cluster C, Condition (2) above is already satisfied.

Case C: $\exists\, C^\star \in \mathcal{C}^\star$**, s.t.,** $C \subset C^\star$. This final case is the most interesting one. We will show that the node it chooses to pair with must be from the same ground truth cluster as C. By condition 1, we know that there must be a C' such that $C' \subseteq C^\star \setminus C$ because the $C \neq C^\star$. There also must exist a C' such that C is connected to C' according to $\phi(\cdot, \cdot)$. Therefore, by the definition of model-based separation C's nearest neighbor will be some cluster that is is connected to and that is in its cluster. Thus we will maintain property 1 in $\mathcal{N}^{(i)}$. ■

5.6.2 Proof of Theorem 8

Theorem 8. *Given a dataset $\boldsymbol{X}$ and a linkage function f such that $\boldsymbol{X}$ is model-based separated with respect to f, let $\mathcal{C}^\star$ be the target partition corresponding to the separated data. Let $\mathcal{D}$ be the DAG-structured clustering produced by* LLAMA *(Alg. 11), then $\mathcal{C}^\star$ is a $\mathcal{D}$ consistent partition, $\mathcal{C}^\star \subset \mathcal{D}$.*

Proof. We will prove this by contradiction and by Lemma 5. Suppose not, let $C^\star \in \mathcal{C}^\star$ be any of the ground truth clusters and $C^\star \notin \mathcal{D}$. In the first round, each member point of $C^\star$ appears as a singleton cluster. Define the pairs of clusters that are nearest neighbors and are both subsets of $C^\star$ in round i and subsequent nodes:

$$\mathcal{N}^{(i)}(C^\star) = \{(C, C') \mid (C, C') \in \mathcal{N}^{(i)},\ C \subset C^\star \wedge C' \subset C^\star\} \tag{5.6}$$

$$\mathcal{S}^{(i)}(C^\star) = \{C \cup C' \mid (C, C') \in \mathcal{N}^{(i)}(C^\star)\} \tag{5.7}$$

Let's consider the earliest round that the above is empty, $\mathcal{N}^{(i)}(C^\star) = \emptyset$, call this round e. We will now show that $C^\star \in \mathcal{S}_{e-1}$. Lemma 1 tells us that each member of the pairs in $\mathcal{N}^{(e-1)}(C^\star)$ must be both subsets $C^\star$ and so we know that: $\forall C'' \in \mathcal{S}^{(e-1)}(C^\star),\ C'' \subseteq C^\star$.

If $\mathcal{N}^{(e)}(C^\star)$ is empty, then we know that for each member $C \in \mathcal{S}^{(e-1)}(C^\star)$ it is the case that C found a nearest neighbor C' such that $C' \not\subset C^\star$. By model based separation, if $C \subset C^\star$ and $C \in \mathcal{S}^{(e-1)}(C^\star)$, then its nearest neighbor must be some other subset that is also a subset of $C^\star$. Some such subset must exist because there exists at least one point that connects the points in C to all other points $C^\star$ in the underlying model-based separation latent graph. And so C must not be a subset of $C^\star$. If this C is not a subset of $C^\star$, then by lemma 1 it must be a superset and by our supposition of $C^\star$ not being in $\mathcal{D}$, a strict superset. But this reaches a contradiction as each member of $\mathcal{S}^{(e-1)}(C^\star)$ was made by merging two pure subsets of $C^\star$. ■

5.6.3 Proof of Proposition 3

Proposition 3 *Given a dataset $\boldsymbol{X}$ and a linkage function f such that $\boldsymbol{X}$ is model-based separated with respect to f, let $\mathcal{C}^\star$ be the target partition corresponding to the noisy model-based separated data. Let $\mathcal{D}$ be the DAG-structured clustering produced by* LLAMA *(Alg. 11), then $\mathcal{C}^\star$ is a $\mathcal{D}$ consistent partition, $\mathcal{C}^\star \subset \mathcal{D}$.*

Proof. The first round of the algorithm creates $\mathcal{N}^{(1)}$ and the clusters that are input to the next round $\mathcal{S}^{(1)}$. We will show that $\mathcal{S}^{(1)}$ is model-based separated with respect to f and the original graph G (not noisy model-based separated) and then apply the results from Theorem 8.

To achieve this, we will show that for each $C^\star \in \mathcal{S}^\star$, $\exists\, C_1, C_2, \ldots, C_K \in \mathcal{S}_1$ such that $\cup_{i=1:k} C_i = C^\star$, there by showing that the original partition $C^\star$ is a model-based separated partition of $\mathcal{S}^{(1)}$. We can partition $\mathcal{S}^{(1)}$ into the connected and disconnected clusters:

$$\mathcal{C}^{(1)}_{\mathsf{conn}} = \{C \,|C \text{ is connected in } G\} \tag{5.8}$$

$$\mathcal{C}^{(1)}_{\mathsf{sep}} = \{C \,|C \text{ is not connected in } G\} \tag{5.9}$$

For each ground truth cluster $C^\star$ the noisy model-based separation property tells us that at most 1/2 of the points of any ground truth cluster can have nearest neighbors that are not connected (and outside the cluster). And so, we have that for each ground truth cluster $C^\star$, each point must participate in at least one member of $\mathcal{S}^{(1)}_{\mathsf{conn}}$, i.e.,

$$\forall C^\star \in \mathcal{C}^\star \; \forall x \in C^\star \; \exists C \in \mathcal{C}^{(1)}_{\mathsf{conn}},\; x \in C \tag{5.10}$$

Now observe that by definition of G, only those ground truth clusters described in the last equation will be connected. For the remaining rounds of the algorithm, all cluster sizes will be greater than 1 and so all nearest neighbors will be from the same cluster, i.e., model-based separation holds. We can use the result from Theorem 8. ■

5.6.4 Proof of Proposition 4

Proposition 4 *There exists a datasets $\boldsymbol{X}$ and symmetric linkage function f such that $\boldsymbol{X}$ is noisy model-based separated wrt f, let $\mathcal{C}^\star$ be the target partition corresponding to the noisy model-based separated data. HAC and* GRINCH *produces a structure $\mathcal{T}$ such that $\mathcal{C}^\star$ is not a tree consistent partition, $\mathcal{C}^\star \not\subset \mathcal{T}$.*

Proof. Consider a very simple dataset with three points $\boldsymbol{X} = \{a, b, c\}$, let the target partition be $\mathcal{C}^\star = \{\{a, b\}, \{c\}\}$. Now let $f(b, c) = 2$ and $f(a, b) = 1$ and $f(a, c) = 0$. We observe that HAC and Grinch will put b and c in the same cluster and so could not represent $\{a, b\}$.

However, we also have that a's nearest neighbor is b and so the DAG-structured method would be able to represent the cluster $\{a, b\}$. ■

5.6.5 Proof of Complexity of LLAMA

Proposition ***(Space Complexity)***. Given a dataset of N points, LLAMA produces DAG-structured clusterings with at most $O(N^2)$ nodes.

Proof. Assuming we have a symmetric linkage function, in each round, each cluster is merged with one other cluster. By the pigeon-hole principle, a round starting with N clusters will produce at most $N - 1$ clusters. Therefore, the total number of nodes that can be produced is $O(\sum_{i=N}^{1} i) = O(N^2)$. ■

Proposition ***(Time Complexity)***. Given a dataset of N points, R rounds of LLAMA produces requires at most $O(R * N^2)$ linkage function computations.

Proof. In each round, we need to find the nearest neighbor of each of the clusters produced by the previous rounds. Without a nearest neighbor index, each round would require $O(N^2)$ time to compute the nearest neighbor of each cluster. If nearest neighbor index structures are used, this time can of course be reduced. ■

Proposition ***(Number of Rounds)***. Let $\mathcal{C}^\star$ be the target partition of a dataset that is (noisy) model-based separated, let K be the size of the largest cluster in $\mathcal{C}^\star$. $K = \max_{C \in \mathcal{C}^\star} |C|$. After K rounds, LLAMA produces a structure that contains $\mathcal{C}^\star$.

Proof. We observe that that all points from the same ground truth cluster will be merged before points from different ground truth clusters. In the worst case, this means that a cluster with K points will take K rounds (by the same logic as the space complexity above) to form. ■

What DAG structures can be formed by LLAMA? We note that the LLAMA algorithm cannot produce any DAG-structured clustering. Instead, it is limited to a subset with

polynomial size. In future work, we hope to better understand the properties of the kind of structure LLAMA can produce.

CHAPTER 6

DISCUSSION AND CONCLUSION

6.1 Summary of Contributions

In this thesis, we have presented algorithms for non-greedy incremental clustering. Our algorithms support constructing tree structures one point at a time while efficiently reconsidering past decisions. We extend our incremental methods to methods that utilize mini-batch parallelism by using bottom-up level-wise approaches. We further extend our tree-based methods to DAG-based methods that represent a richer class of nested, overlapping clusterings. We analyze each method that is presented both theoretically and empirically. Theoretically, we study a generalization of commonly used clustering assumptions which we call model-based separation. Empirically, we evaluate on both clustering benchmarks as well as entity resolution tasks. We achieve multiple point improvements in dendrogram purity and scalability to billions of points.

6.2 Limitations

There are several limitations to the work presented in this thesis. First, we note that all methods presented scale as a function of the dataset size. This could become prohibitively expensive for very large datasets. Further, for certain applications it is not needed to give the cluster assignment of every point, but rather the discovered cluster parameters (e.g., centroid) are of interest. Coreset based methods for clustering could be considered to scale independently of the dataset size.

Second, we did not focus on how to efficiently operate in distributed computing frameworks. However, the algorithms presented have sub-components with a long history of

distributed computing (e.g., connected components, nearest neighbor search). It would be important to be better able to determine where and when distributed computation is necessary and how to modify off-the-shelf distributed algorithms conditioned on their use in our proposed algorithms.

Third, we note that much of the running time complexity of the algorithms is in the nearest neighbor search operations. In one sense, this makes the algorithms modular, with each advancement in nearest neighbor search techniques improving our methods. However, it often requires the construction of an auxiliary structure for nearest neighbor search. It would be interesting to instead consider how search and clustering can use the same structure.

Fourth, the analysis of algorithms is typically with respect to separability assumptions of the data. It would be important to better understand the degree to which these assumptions hold in practice. We also did not provide analysis with respect to hierarchical clustering costs such as Dasgupta's [94].

Lastly, we assume a predefined similarity function is given for all of the proposed methods. Often this function would not be known in advance and we would like to jointly discover the clustering and fit this similarity function.

6.3 Future Work

In this thesis, we explored building DAG-structured clusterings in the batch setting where the entire dataset is known in advance (Chapter 5). We can extend the LLAMA algorithm (Algorithm 11) for DAG construction to give a mini-batch algorithm in the same way MBSCC (Algorithm 10) extends SCC (Algorithm 9). This algorithm would simply update the 1-nearest neighbor selection of each node in the DAG structure. Beyond this simple extension, there are several interesting considerations of DAG structured clustering in the online setting. It would be interesting to consider how to use the DAG structure to represent a distribution over hierarchical clusterings in the online setting [134]. Representing uncertainty over hierarchical clusterings could provide a mechanism to recover from greediness /

difficulties of operating in the online setting in a way similar to beam search or Combinatorial Sequential Monte Carlo [212, 279]. We could also explore more compact DAG-structured representations of multiple trees [134] and how to search [133] through these structures for high quality tree structures at the scale of work presented in this thesis and in the online setting. This could include continuous relaxations of tree and DAG-structures [64].

A strong assumption in this thesis is the use of predefined vector representation of similarity function. However, in many cases these representations are not known in advance, but rather we would want to jointly learn representations and perform clustering. Recent work [62] has shown the benefits of using clustering as a part of the representation learning objective. It would be interesting to consider how the kinds of online clustering methods and DAG-structured clustering methods that are central to this thesis could be used in the representation learning setting. It would also be interested to consider these clustering approaches in grouped-data setting such as language models discovering word and topic embeddings [294].

Most of the applications considered in this thesis are focused on clustering for data analysis or entity resolution. However, there several other tasks and downstream uses cases that could be interesting applications of our work.

In reinforcement learning, the goal is to maximize the expected discounted reward attained by an agent. Q-learning builds a function which estimates the expected reward of each action in each state. Deep learning methods [208] often require a large amount of training data to achieve high quality results. To reduce this complexity model-free episodic control [46, 234], replaces the complex deep model with a simple non-parametric approach. It does this by not having a look up table that uses k-nearest neighbor to generalize to unobserved states. We could replace the k-nearest neighbor based approach for episodic control with latent variable approach. The latent variable approach will assign states to clusters and model the reward of a cluster-action pair. We hope that this will allow for better

generalization as well as a smaller Q table. Our online clustering work would be used to incrementally update the assignment of states to clusters.

Nearest neighbor indices are often used for efficient retrieval models in larger neural network architectures [138]. For instance, in question answering, candidate paragraphs among a massive corpora which may contain the answer [91] to a particular query. Often, the model that is used to do retrieval is based on a *learned* encoding of the query and the candidate paragraphs. This model embeds each query and each candidate paragraph as a vector to facilitate efficient nearest neighbor computations using the aforementioned index structures. This learned model is updated during training, thus changing the embedding vectors for each query and candidate paragraph. This is an expensive operation and we would like to consider ways to approximate this effectively. We could perhaps provide an approximation based on maintaining a clustering of the data and only updating the encoded representation of a point if another point within the same cluster was observed in training.

In continual learning and streaming settings, we attempt to design algorithms for efficient training classification models in the presence of ever-changing training data distributions. In particular, we hope to design methods to avoid catastrophic forgetting [200]. Recent work has shown *replay-memory* based approaches which summarize past training data with a subset of examples and continue to train on these summaries in the presence of new data are quite effective [48, 193, 268]. Recently work has used coresets, which are akin to fine-grained clusters, to determine these training data summaries for continual learning [48]. We could potentially use our hierarchical and DAG-structured clustering methods to represent multiple granularities of coresets for continual learning. We hope that this structure would provide for more useful abstractions as it allows for more uncertainty and relaxes a hard-assignment constraint on points to coresets.

Lastly, we could consider ways to do representation learning / similarity function learning not just from one dataset, but from a collection of datasets. We observed in our work that the linkage function of average linkage restricted to include only the top-K nearest neighbors is

very effective across all datasets. This raises an interesting question as to how to know the number of neighbors that is best for a given dataset. Furthermore, it raises questions about whether *higher* order neighbor relationships should be considered, such as which points are the nearest neighbor of the nearest neighbors of a given point. Rather than trying to design a model ourselves to capture these properties, we propose to use labeled data (e.g., classification datasets) to *learn* what inter/intra-cluster neighborhoods look like. Dataset agnostic features using K-nn graphs with edges labeled by *percentile* or *rank* of the similarity rather than the individual features of the data point can be used. One could then train a model to weight or embed these features in a manner such that top-K average linkage is an effective clustering algorithm using training techniques such as those in our previous work on supervised clustering [288]. This model has the potential to discover patterns of how clusters are formed across an array of different datasets, which we hope would be beneficial to clustering unobserved datasets.

BIBLIOGRAPHY

[1] Abboud, Amir, Cohen-Addad, Vincent, and Houdrougé, Hussein. Subquadratic high-dimensional hierarchical clustering. *Advances in Neural Information Processing Systems (NeurIPS)* (2019).

[2] Acar, Umut A., Anderson, Daniel, Blelloch, Guy E., and Dhulipala, Laxman. Parallel batch-dynamic graph connectivity. *Symposium on Parallelism in Algorithms and Architectures (SPAA)* (2019).

[3] Acar, Umut A., Anderson, Daniel, Blelloch, Guy E., Dhulipala, Laxman, and Westrick, Sam. Parallel batch-dynamic trees via change propagation. *European Symposium on Algorithms (ESA)* (2020).

[4] Acar, Umut A., Blelloch, Guy E., and Vittes, Jorge L. An experimental analysis of change propagation in dynamic trees. *The Algorithm Engineering and Experiments (ALENEX) and Analytic Algorithmics and Combinatorics (ANALCO)* (2005).

[5] Acar, Umut A., Cotter, Andrew, Hudson, Benoît, and Türkoglu, Duru. Dynamic well-spaced point sets. *Symposium on Computational Geometry (SoCG)* (2010).

[6] Acar, Umut A., Cotter, Andrew, Hudson, Benoît, and Türkoglu, Duru. Parallelism in dynamic well-spaced point sets. *Symposium on Parallelism in Algorithms and Architectures (SPAA)* (2011).

[7] Achtert, Elke, Böhm, Christian, Kriegel, Hans-Peter, Kröger, Peer, and Zimek, Arthur. Robust, complete, and efficient correlation clustering. *International Conference on Data Mining (ICDM)* (2007).

[8] Ackermann, Marcel R, Märtens, Marcus, Raupach, Christoph, Swierkot, Kamil, Lammersen, Christiane, and Sohler, Christian. Streamkm++ a clustering algorithm for data streams. *Journal of Experimental Algorithmics (JEA)* (2012).

[9] Aggarwal, Charu C, and Reddy, Chandan K. Data clustering, 2014.

[10] Ahmadian, Sara, Epasto, Alessandro, Kumar, Ravi, and Mahdian, Mohammad. Fair correlation clustering. *Artificial Intelligence and Statistics (AISTATS)* (2020).

[11] Ahmed, Amr, Ravi, Sujith, Smola, Alex, and Narayanamurthy, Shravan. Fastex: Hash clustering with exponential families. *Advances in Neural Information Processing Systems (NeurIPS)* (2012).

[12] Ahn, KookJin, Cormode, Graham, Guha, Sudipto, McGregor, Andrew, and Wirth, Anthony. Correlation clustering in data streams. *International Conference on Machine Learning (ICML)* (2015).

[13] Ailon, Nir, and Charikar, Moses. Fitting tree metrics: Hierarchical clustering and phylogeny. *Foundations of Computer Science (FOCS)* (2005).

[14] Ailon, Nir, Charikar, Moses, and Newman, Alantha. Aggregating inconsistent information: Ranking and clustering. *Journal of the ACM (JACM)* (2008).

[15] Ailon, Nir, Jaiswal, Ragesh, and Monteleoni, Claire. Streaming k-means approximation. *Advances in Neural Information Processing Systems (NeurIPS)* (2009).

[16] Ailon, Nir, and Liberty, Edo. Correlation clustering revisited: The "true" cost of error minimization problems. *International Colloquium on Automata, Languages, and Programming (ICALP)* (2009).

[17] Alstrup, Stephen, Holm, Jacob, de Lichtenberg, Kristian, and Thorup, Mikkel. Maintaining information in fully dynamic trees with top trees. *ACM Transactions on Algorithms (TALG)* (2005).

[18] Anderberg, Michael R. Cluster analysis for applications, 1973.

[19] Andrews, Nicholas, Eisner, Jason, and Dredze, Mark. Robust entity clustering via phylogenetic inference. *Association for Computational Linguistics (ACL)* (2014).

[20] Ankerst, Mihael, Breunig, Markus M., Kriegel, Hans-Peter, and Sander, Jörg. Ordering points to identify the clustering structure. *SIGMOD* (2008).

[21] Awasthi, Pranjal, Blum, Avrim, and Sheffet, Or. Center-based clustering under perturbation stability. *Information Processing Letters* (2012).

[22] Bachem, Olivier, Lucic, Mario, Hassani, Hamed, and Krause, Andreas. Fast and provably good seedings for k-means. *Advances in Neural Information Processing Systems (NeurIPS)* (2016).

[23] Bachem, Olivier, Lucic, Mario, and Krause, Andreas. Coresets for nonparametric estimation-the case of dp-means. *International Conference on Machine Learning (ICML)* (2015).

[24] Bachem, Olivier, Lucic, Mario, and Krause, Andreas. Distributed and provably good seedings for k-means in constant rounds. *International Conference on Machine Learning (ICML)* (2017).

[25] Balakrishnan, Sivaraman, Xu, Min, Krishnamurthy, Akshay, and Singh, Aarti. Noise thresholds for spectral clustering. *Advances in Neural Information Processing Systems (NeurIPS)* (2011).

[26] Balcan, Maria-Florina, Blum, Avrim, and Vempala, Santosh. A discriminative framework for clustering via similarity functions. *Symposium on Theory of Computing (STOC)* (2008).

[27] Balcan, Maria-Florina, Haghtalab, Nika, and White, Colin. k-center clustering under perturbation resilience. *ACM Transactions on Algorithms (TALG)* (2020).

[28] Balcan, Maria-Florina, and Liang, Yingyu. Clustering under perturbation resilience. *SIAM Journal on Computing* (2016).

[29] Balcan, Maria-Florina, Liang, Yingyu, and Gupta, Pramod. Robust hierarchical clustering. *Journal of Machine Learning Research (JMLR)* (2014).

[30] Bansal, Nikhil, Blum, Avrim, and Chawla, Shuchi. Correlation clustering. *Machine learning* (2004).

[31] Bateni, Mohammadhossein, Behnezhad, Soheil, Derakhshan, Mahsa, Hajiaghayi, MohammadTaghi, Kiveris, Raimondas, Lattanzi, Silvio, and Mirrokni, Vahab. Affinity clustering: Hierarchical clustering at scale. *Advances in Neural Information Processing Systems (NeurIPS)* (2017).

[32] Benevenuto, Fabrício, Rodrigues, Tiago, Cha, Meeyoung, and Almeida, Virgílio. Characterizing user navigation and interactions in online social networks. *Information Science* (2012).

[33] Bertrand, Patrice, and Diatta, Jean. Multilevel clustering models and interval convexities. *Discrete Applied Mathematics* (2017).

[34] Bertrand, Patrice, and Janowitz, Melvin F. Pyramids and weak hierarchies in the ordinal model for clustering. *Discrete Applied Mathematics* (2002).

[35] Betancourt, Brenda, Sosa, Juan, and Rodríguez, Abel. A prior for record linkage based on allelic partitions. *arXiv preprint* (2020).

[36] Betancourt, Brenda, Zanella, Giacomo, Miller, Jeffrey W, Wallach, Hanna, Zaidi, Abbas, and Steorts, Rebecca C. Flexible models for microclustering with application to entity resolution. *Advances in Neural Information Processing Systems (NeurIPS)* (2016).

[37] Betancourt, Brenda, Zanella, Giacomo, and Steorts, Rebecca C. Random partition models for microclustering tasks. *Journal of the American Statistical Association* (2020).

[38] Beygelzimer, Alina, Kakade, Sham, and Langford, John. Cover trees for nearest neighbor. *International Conference on Machine Learning (ICML)* (2006).

[39] Bhaskara, Aditya, and Rwanpathirana, Aravinda Kanchana. Robust algorithms for online k-means clustering. *Algorithmic Learning Theory (ALT)* (2020).

[40] Bhattacharjee, Robi, and Moshkovitz, Michal. No-substitution k-means clustering with adversarial order. *Algorithmic Learning Theory (ALT)* (2021).

[41] Bilu, Yonatan, and Linial, Nathan. Are stable instances easy? *Combinatorics, Probability and Computing* (2012).

[42] Binette, Olivier, and Steorts, Rebecca C. Some of entity resolution, 2021.

[43] Blei, David M, Griffiths, Thomas L, and Jordan, Michael I. The nested chinese restaurant process and Bayesian nonparametric inference of topic hierarchies. *Journal of the ACM (JACM)* (2010).

[44] Blundell, Charles, and Teh, Yee Whye. Bayesian hierarchical community discovery. *Advances in Neural Information Processing Systems (NeurIPS)* (2013).

[45] Blundell, Charles, Teh, Yee Whye, and Heller, Katherine A. Bayesian rose trees. *Conference on Uncertainty in Artificial Intelligence (UAI)* (2010).

[46] Blundell, Charles, Uria, Benigno, Pritzel, Alexander, Li, Yazhe, Ruderman, Avraham, Leibo, Joel Z, Rae, Jack, Wierstra, Daan, and Hassabis, Demis. Model-free episodic control. *arXiv Pre-print* (2016).

[47] Bo, Deyu, Wang, Xiao, Shi, Chuan, Zhu, Meiqi, Lu, Emiao, and Cui, Peng. Structural deep clustering network. *The Web Conference (WWW)* (2020).

[48] Borsos, Zalán, Mutnỳ, Mojmír, and Krause, Andreas. Coresets via bilevel optimization for continual learning and streaming. *Advances in Neural Information Processing Systems (NeurIPS)* (2020).

[49] Borůvka, Otakar. O jistém problému minimálním, 1926.

[50] Bottou, Leon, and Bengio, Yoshua. Convergence properties of the k-means algorithms. *Advances in Neural Information Processing Systems (NeurIPS)* (1995).

[51] Boyles, Levi, and Welling, Max. The time-marginalized coalescent prior for hierarchical clustering. *Advances in Neural Information Processing Systems (NeurIPS)* (2012).

[52] Bradley, Paul, Mangasarian, Olvi, and Street, W. Clustering via concave minimization. *Advances in Neural Information Processing Systems (NeurIPS)* (1996).

[53] Broderick, Tamara, Kulis, Brian, and Jordan, Michael. Mad-bayes: Map-based asymptotic derivations from bayes. *International Conference on Machine Learning (ICML)* (2013).

[54] Brown, Peter F, Desouza, Peter V, Mercer, Robert L, Pietra, Vincent J Della, and Lai, Jenifer C. Class-based n-gram models of natural language. *Computational linguistics* (1992).

[55] Bruynooghe, Michel. Classification ascendante hiérarchique des grands ensembles de données: un algorithme rapide fondé sur la construction des voisinages réductibles. *Cahiers de l'analyse des données* (1978).

[56] Campello, Ricardo J. G. B., Moulavi, Davoud, and Sander, Joerg. Density-based clustering based on hierarchical density estimates. *Advances in Knowledge Discovery and Data Mining* (2013).

[57] Cao, Feng, Estert, Martin, Qian, Weining, and Zhou, Aoying. Density-based clustering over an evolving data stream with noise. *International Conference on Data Mining (ICDM)* (2006).

[58] Cao, Nan, Sun, Jimeng, Lin, Yu-Ru, Gotz, David, Liu, Shixia, and Qu, Huamin. Facetatlas: Multifaceted visualization for rich text corpora. *IEEE transactions on visualization and computer graphics* (2010).

[59] Carlsson, Gunnar, Mémoli, Facundo, Ribeiro, Alejandro, and Segarra, Santiago. Hierarchical Quasi-Clustering methods for asymmetric networks. *International Conference on Machine Learning (ICML)* (2014).

[60] Carlsson, Gunnar E, and Mémoli, Facundo. Characterization, stability and convergence of hierarchical clustering methods. *Journal of Machine Learning Research (JMLR)* (2010).

[61] Caron, Mathilde, Bojanowski, Piotr, Joulin, Armand, and Douze, Matthijs. Deep clustering for unsupervised learning of visual features. *European Conference on Computer Vision (ECCV)* (2018).

[62] Caron, Mathilde, Misra, Ishan, Mairal, Julien, Goyal, Priya, Bojanowski, Piotr, and Joulin, Armand. Unsupervised learning of visual features by contrasting cluster assignments. *Advances in Neural Information Processing Systems (NeurIPS)* (2020).

[63] Cattan, Arie, Eirew, Alon, Stanovsky, Gabriel, Joshi, Mandar, and Dagan, Ido. Streamlining cross-document coreference resolution: Evaluation and modeling. *arXiv Pre-print* (2020).

[64] Chami, Ines, Gu, Albert, Chatziafratis, Vaggos, and R'e, Christopher. From trees to continuous embeddings and back: Hyperbolic hierarchical clustering. *Advances in Neural Information Processing Systems (NeurIPS)* (2020).

[65] Chang, Jianlong, Wang, Lingfeng, Meng, Gaofeng, Xiang, Shiming, and Pan, Chunhong. Deep adaptive image clustering. *International Conference on Computer Vision (ICCV)* (2017).

[66] Charikar, M., Chekuri, C., Feder, T., and Motwani, R. Incremental clustering and dynamic information retrieval. *Symposium on Theory of Computing (STOC)* (1997).

[67] Charikar, M., Guruswami, V., and Wirth, A. Clustering with qualitative information. *Foundations of Computer Science (FOCS)* (2003).

[68] Charikar, Moses, and Chatziafratis, Vaggos. Approximate hierarchical clustering via sparsest cut and spreading metrics. *Symposium on Discrete Algorithms (SODA)* (2017).

[69] Charikar, Moses, Chatziafratis, Vaggos, and Niazadeh, Rad. Hierarchical clustering better than average-linkage. *Symposium on Discrete Algorithms (SODA)* (2019).

[70] Charikar, Moses, Chatziafratis, Vaggos, Niazadeh, Rad, and Yaroslavtsev, Grigory. Hierarchical clustering for euclidean data. *Artificial Intelligence and Statistics (AISTATS)* (2019).

[71] Charikar, Moses, O'Callaghan, Liadan, and Panigrahy, Rina. Better streaming algorithms for clustering problems. *Symposium on Theory of Computing (STOC)* (2003).

[72] Chaudhuri, Kamalika, and Dasgupta, Sanjoy. Rates of convergence for the cluster tree. *Advances in Neural Information Processing Systems (NeurIPS)* (2010).

[73] Chen, Beidi, Shrivastava, Anshumali, Steorts, Rebecca C, et al. Unique entity estimation with application to the syrian conflict. *The Annals of Applied Statistics* (2018).

[74] Chen, Ke. On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications. *SIAM Journal on Computing* (2009).

[75] Chen, Qingyu, Peng, Yifan, and Lu, Zhiyong. Biosentvec: creating sentence embeddings for biomedical texts. *International Conference on Healthcare Informatics (ICHI)* (2019).

[76] Cheng, Yizong. Mean shift, mode seeking, and clustering. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (1995).

[77] Cheng, Yizong, and Fu, King-Sun. Conceptual clustering in knowledge organization. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (1985).

[78] Chierichetti, Flavio, Dalvi, Nilesh, and Kumar, Ravi. Correlation clustering in mapreduce. *Conference on Knowledge Discovery and Data Mining (KDD)* (2014).

[79] Choromanska, Anna, and Monteleoni, Claire. Online clustering with experts. *Artificial Intelligence and Statistics (AISTATS)* (2012).

[80] Chung, Fan RK, and Graham, Fan Chung. Spectral graph theory, 1997.

[81] Clark, Kevin, and Manning, Christopher D. Improving coreference resolution by learning entity-level distributed representations. *Association for Computational Linguistics (ACL)* (2016).

[82] Cohen-Addad, Vincent, Guedj, Benjamin, Kanade, Varun, and Rom, Guy. Online k-means clustering. *Artificial Intelligence and Statistics (AISTATS)* (2021).

[83] Cohen-Addad, Vincent, Kanade, Varun, Mallmann-Trenn, Frederik, and Mathieu, Claire. Hierarchical clustering: Objective functions and algorithms. *Symposium on Discrete Algorithms (SODA)* (2018).

[84] Cohen-Addad, Vincent, Karthik, CS, and Lagarde, Guillaume. On efficient low distortion ultrametric embedding. *International Conference on Machine Learning (ICML)* (2020).

[85] Collins, Christopher, Carpendale, Sheelagh, and Penn, Gerald. Docuburst: Visualizing document content using language structure. *Computer graphics forum* (2009).

[86] Comaniciu, Dorin, and Meer, Peter. Mean shift: A robust approach toward feature space analysis. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2002).

[87] Cranmer, Kyle, Macaluso, Sebastian, and Pappadopulo, Duccio. Toy generative model for jets, 2019.

[88] Culbertson, Jared, Guralnik, Dan P, and Stiller, Peter F. Functorial hierarchical clustering with overlaps. *Discrete Applied Mathematics* (2018).

[89] Culotta, A., Kanani, P., Hall, R., Wick, M., and McCallum, A. Author disambiguation using error-driven machine learning with a ranking loss function. *Workshop on Information Integration on the Web* (2007).

[90] Curtin, Ryan R. Dual-tree k-means with bounded iteration runtime. *arXiv Pre-print* (2016).

[91] Das, Rajarshi, Dhuliawala, Shehzaad, Zaheer, Manzil, and McCallum, Andrew. Multi-step retriever-reader interaction for scalable open-domain question answering. *International Conference on Learning Representations (ICLR)* (2019).

[92] Das, Rajarshi, Godbole, Ameya, Monath, Nicholas, Zaheer, Manzil, and McCallum, Andrew. Probabilistic case-based reasoning for open-world knowledge graph completion. *Findings of the Association for Computational Linguistics: EMNLP* (2020).

[93] Dasgupta, Sanjoy. The hardness of k-means clustering, 2008.

[94] Dasgupta, Sanjoy. A cost function for similarity-based hierarchical clustering. *Symposium on Theory of Computing (STOC)* (2016).

[95] Day, William H. E., and Edelsbrunner, Herbert. Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of Classification* (1984).

[96] Defays, D. An efficient algorithm for a complete link method. *The Computer Journal* (1977).

[97] Dell'Amico, Matteo. Fishdbc: Flexible, incremental, scalable, hierarchical density-based clustering for arbitrary data and distance. *arXiv Pre-print* (2019).

[98] Demaine, Erik D, and Immorlica, Nicole. Correlation clustering with partial information. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques* (2003).

[99] Devlin, Jacob, Chang, Ming-Wei, Lee, Kenton, and Toutanova, Kristina. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv Pre-print* (2018).

[100] Dhillon, Inderjit S, Guan, Yuqiang, and Kulis, Brian. Kernel k-means: spectral clustering and normalized cuts. *Conference on Knowledge Discovery and Data Mining (KDD)* (2004).

[101] Dhulipala, Laxman, Eisenstat, David, Łącki, Jakub, Mirrokni, Vahab, and Shi, Jessica. Hierarchical agglomerative graph clustering in nearly-linear time. *International Conference on Machine Learning (ICML)* (2021).

[102] Diday, Edwin. *Orders and overlapping clusters by pyramids*. PhD thesis, INRIA, 1987.

[103] Ding, Yufei, Zhao, Yue, Shen, Xipeng, Musuvathi, Madanlal, and Mytkowicz, Todd. Yinyang k-means: A drop-in replacement of the classic k-means with consistent speedup. *International Conference on Machine Learning (ICML)* (2015).

[104] Dredze, Mark, Andrews, Nicholas, and DeYoung, Jay. Twitter at the grammys: A social media corpus for entity linking and disambiguation. *SocialNLP@EMNLP* (2016).

[105] Dubey, Avinava, Zhang, Michael Minyi, Xing, Eric P, and Williamson, Sinead A. Distributed, partially collapsed mcmc for bayesian nonparametrics. *Artificial Intelligence and Statistics (AISTATS)* (2020).

[106] Dubey, Kumar Avinava, Ho, Qirong, Williamson, Sinead A, and Xing, Eric P. Dependent nonparametric trees for dynamic hierarchical clustering. *Advances in Neural Information Processing Systems (NeurIPS)* (2014).

[107] Ein Dor, Liat, Mass, Yosi, Halfon, Alon, Venezian, Elad, Shnayderman, Ilya, Aharonov, Ranit, and Slonim, Noam. Learning thematic similarity metric from article sections using triplet networks. *Association for Computational Linguistics (ACL)* (2018).

[108] Eisen, M. B., Spellman, P. T., Brown, P. O., and Botstein, D. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences* (1998).

[109] Elkan, Charles. Using the triangle inequality to accelerate k-means. *International Conference on Machine Learning (ICML)* (2003).

[110] Epasto, Alessandro, Lattanzi, Silvio, and Paes Leme, Renato. Ego-splitting framework: From non-overlapping to overlapping clusters. *Conference on Knowledge Discovery and Data Mining (KDD)* (2017).

[111] Eriksson, Brian, Dasarathy, Gautam, Singh, Aarti, and Nowak, Rob. Active clustering: Robust and efficient hierarchical clustering using adaptively selected similarities. *Artificial Intelligence and Statistics (AISTATS)* (2011).

[112] Ester, Martin, Kriegel, Hans-Peter, Sander, Jörg, Xu, Xiaowei, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. *Conference on Knowledge Discovery and Data Mining (KDD)* (1996).

[113] Fellbaum, Christiane. Wordnet: An electronic lexical database., 1998.

[114] Fichtenberger, Hendrik, Gillé, Marc, Schmidt, Melanie, Schwiegelshohn, Chris, and Sohler, Christian. Bico: Birch meets coresets for k-means clustering. *European Symposium on Algorithms (ESA)* (2013).

[115] Forgey, Edward. Cluster analysis of multivariate data: Efficiency vs. interpretability of classification. *Biometrics* (1965).

[116] Fowlkes, Charless, Belongie, Serge, Chung, Fan, and Malik, Jitendra. Spectral grouping using the nystrom method. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2004).

[117] Frederickson, Greg N. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM Journal on Computing* (1985).

[118] Frey, Brendan J, and Dueck, Delbert. Clustering by passing messages between data points. *Science* (2007).

[119] Fukunaga, Keinosuke, and Hostetler, Larry. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory* (1975).

[120] Gama, Fernando, Segarra, Santiago, and Ribeiro, Alejandro. Hierarchical overlapping clustering of network data using cut metrics. *IEEE Transactions on Signal and Information Processing over Networks* (2017).

[121] Garg, Ashwani, Mangla, Ashish, Gupta, Neelima, and Bhatnagar, Vasudha. Pbirch: A scalable parallel clustering algorithm for incremental data. *International Database Engineering and Applications Symposium (IDEAS)* (2006).

[122] Gavryushkin, Alex, and Drummond, Alexei J. The space of ultrametric phylogenetic trees. *Journal of theoretical biology* (2016).

[123] Geusebroek, J., Burghouts, G. J., and Smeulders, A. W.M. The amsterdam library of object images. *International Journal of Computer Vision (IJCV)* (2005).

[124] Ghahramani, Zoubin, Jordan, Michael I, and Adams, Ryan P. Tree-structured stick breaking for hierarchical data. *Advances in Neural Information Processing Systems (NeurIPS)* (2010).

[125] Gilbert, Seth, and Li, Lawrence Er Lu. How fast can you update your mst? *Symposium on Parallelism in Algorithms and Architectures (SPAA)* (2020).

[126] Gilpin, Sean, Nijssen, Siegried, and Davidson, Ian. Formalizing hierarchical clustering as integer linear programming. *AAAI Conference on Artificial Intelligence (AAAI)* (2013).

[127] Gonzalez, Joseph, Low, Yucheng, Gretton, Arthur, and Guestrin, Carlos. Parallel gibbs sampling: From colored fields to thin junction trees. *Artificial Intelligence and Statistics (AISTATS)* (2011).

[128] Gonzalez, Teofilo F. Clustering to minimize the maximum intercluster distance. *Theoretical computer science* (1985).

[129] Gower, John C., and Ross, Gavin J. S. Minimum spanning trees and single linkage cluster analysis. *Journal of The Royal Statistical Society Series C-applied Statistics* (1969).

[130] Goyal, Prasoon, Hu, Zhiting, Liang, Xiaodan, Wang, Chenyu, Xing, Eric P., and Mellon, Carnegie. Nonparametric variational auto-encoders for hierarchical representation learning. *International Conference on Computer Vision (ICCV)* (2017).

[131] Green, Spence, Andrews, Nicholas, Gormley, Matthew R., Dredze, Mark, and Manning, Christopher D. Entity clustering across languages. *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)* (2012).

[132] Greenberg, C. S., Bansé, D., Doddington, G. R., Garcia-Romero, D., Godfrey, J. J., Kinnunen, T., Martin, A. F., McCree, A., Przybocki, M., and Reynolds, D. A. The nist 2014 speaker recognition i-vector machine learning challenge. *Odyssey: The Speaker and Language Recognition Workshop* (2014).

[133] Greenberg, Craig S., Macaluso, Sebastian, Monath, Nicholas, Dubey, Kumar Avinava, Flaherty, Patrick, Zaheer, Manzil, Ahmed, Amr, Cranmer, Kyle, and McCallum, Andrew. Exact and approximate hierarchical clustering using A*. *Conference on Uncertainty in Artificial Intelligence (UAI)* (2021).

[134] Greenberg, Craig S., Macaluso, Sebastian, Monath, Nicholas, Lee, Ji-Ah, Flaherty, Patrick, Cranmer, Kyle, McGregor, Andrew, and McCallum, Andrew. Cluster trellis: Data structures and algorithms for exact inference in hierarchical clustering, 2021.

[135] Griffiths, Thomas L, and Ghahramani, Zoubin. The indian buffet process: An introduction and review. *Journal of Machine Learning Research (JMLR)* (2011).

[136] Großwendt, Anna, Röglin, Heiko, and Schmidt, Melanie. Analysis of ward's method. *Symposium on Discrete Algorithms (SODA)* (2019).

[137] Guo, Ruiqi, Sun, Philip, Lindgren, Erik, Geng, Quan, Simcha, David, Chern, Felix, and Kumar, Sanjiv. Accelerating large-scale inference with anisotropic vector quantization. *International Conference on Machine Learning (ICML)* (2020).

[138] Guu, Kelvin, Lee, Kenton, Tung, Zora, Pasupat, Panupong, and Chang, Ming-Wei. Realm: Retrieval-augmented language model pre-training. *arXiv Pre-print* (2020).

[139] Haghighi, Aria, and Klein, Dan. Coreference resolution in a modular, entity-centered model. *Association for Computational Linguistics: Human Language Technologies (ACL-HLT)* (2010).

[140] Hakimi, S Louis. Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations research* (1964).

[141] Han, Hui, Zha, Hongyuan, and Giles, C. Lee. Name disambiguation in author citations using a k-way spectral clustering method. *Joint Conference on Digital Libraries (JCDL)* (2005).

[142] Har-Peled, Sariel, and Mazumdar, Soham. On coresets for k-means and k-median clustering. *Symposium on Theory of Computing (STOC)* (2004).

[143] Har-Peled, Sariel, and Sadri, Bardia. How fast is the k-means method? *Algorithmica* (2005).

[144] Hartigan, John A. Clustering algorithms, 1975.

[145] Heller, K, and Ghahramani, Zoubin. Randomized algorithms for fast bayesian hierarchical clustering. *EU-PASCAL Statistics and Optimization of Clustering Workshop* (2005).

[146] Heller, Katherine A, and Ghahramani, Zoubin. Bayesian hierarchical clustering. *International Conference on Machine Learning (ICML)* (2005).

[147] Henzinger, Monika, and King, Valerie. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *Journal of the ACM (JACM)* (1999).

[148] Henzinger, Monika, and King, Valerie. Maintaining minimum spanning forests in dynamic graphs. *SIAM Journal on Computing* (2001).

[149] Hoffman, Matthew D, Blei, David M, Wang, Chong, and Paisley, John. Stochastic variational inference. *Journal of Machine Learning Research (JMLR)* (2013).

[150] Holm, Jacob, de Lichtenberg, Kristian, and Thorup, Mikkel. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM (JACM)* (2001).

[151] Hu, Yuening, Ying, Jordan L, Daume III, Hal, and Ying, Z. Irene. Binary to bushy: Bayesian hierarchical clustering with the beta coalescent. *Advances in Neural Information Processing Systems (NeurIPS)* (2013).

[152] Hu, Zhiting, Qirong, Ho, Dubey, Avinava, and Xing, Eric. Large-scale distributed dependent nonparametric trees. *International Conference on Machine Learning (ICML)* (2015).

[153] Jain, Anil K, and Dubes, Richard C. Algorithms for clustering data, 1988.

[154] Jang, Jennifer, and Jiang, Heinrich. Dbscan++: Towards fast and scalable density clustering. *International Conference on Machine Learning (ICML)* (2019).

[155] Jardine, Nicholas, and Sibson, Robin. Mathematical taxonomy, 1971.

[156] Jeantet, Ian, Miklós, Zoltán, and Gross-Amblard, David. Overlapping hierarchical clustering (OHC). *Advances in Intelligent Data Analysis* (2020).

[157] Jiang, Ke, Kulis, Brian, and Jordan, Michael I. Small-variance asymptotics for exponential family Dirichlet process mixture models. *Advances in Neural Information Processing Systems (NeurIPS)* (2012).

[158] Jiang, Zhuxi, Zheng, Yin, Tan, Huachun, Tang, Bangsheng, and Zhou, Hanning. Variational deep embedding: A generative approach to clustering. *arXiv Pre-print* (2016).

[159] Jin, Chen, Liu, Ruoqian, Chen, Zhengzhang, Hendrix, William, Agrawal, Ankit, and Choudhary, Alok. A scalable hierarchical clustering algorithm using spark. *International Conference on Big Data Computing Service and Applications* (2015).

[160] Jin, Chen, Patwary, Md Mostofa Ali, Agrawal, Ankit, Hendrix, William, Liao, Wei-keng, and Choudhary, Alok. Disc: A distributed single-linkage hierarchical clustering algorithm using mapreduce. *Symposium on High Performance Computing (HPC)* (2015).

[161] Johnson, Jeff, Douze, Matthijs, and Jégou, Hervé. Billion-scale similarity search with gpus. *arXiv Pre-print* (2017).

[162] Kalhan, Sanchit, Makarychev, Konstantin, and Zhou, Timothy. Correlation clustering with local objectives. *Advances in Neural Information Processing Systems (NeurIPS)* (2019).

[163] Kenyon-Dean, Kian, Cheung, Jackie Chi Kit, and Precup, Doina. Resolving event coreference with supervised representation learning and clustering-oriented regularization. *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics* (2018).

[164] Kim, Jaeyoung, Yoon, Janghyeok, Park, Eunjeong, and Choi, Sungchul. Patent document clustering with deep embeddings. *Scientometrics* (2020).

[165] Kiveris, Raimondas, Lattanzi, Silvio, Mirrokni, Vahab, Rastogi, Vibhor, and Vassilvitskii, Sergei. Connected components in mapreduce and beyond. *SOCC* (2014).

[166] Kleinberg, Jon. The small-world phenomenon: An algorithmic perspective. *Symposium on Theory of Computing (STOC)* (2000).

[167] Kleinberg, Jon. An impossibility theorem for clustering. *Advances in Neural Information Processing Systems (NeurIPS)* (2002).

[168] Kleinberg, Jon. Complex networks and decentralized search algorithms. *International Congress of Mathematicians (ICM)* (2006).

[169] Knowles, David A, and Ghahramani, Zoubin. Pitman-yor diffusion trees. *Conference on Uncertainty in Artificial Intelligence (UAI)* (2011).

[170] Kobren, Ari, Monath, Nicholas, Krishnamurthy, Akshay, and McCallum, Andrew. A hierarchical algorithm for extreme clustering. *Conference on Knowledge Discovery and Data Mining (KDD)* (2017).

[171] Kobren, Ari, Monath, Nicholas, and McCallum, Andrew. Integrating user feedback under identity uncertainty in knowledge base construction. *Automated Knowledge Base Construction (AKBC)* (2019).

[172] Kohli, Pushmeet, Torr, Philip HS, et al. Robust higher order potentials for enforcing label consistency. *International Journal of Computer Vision (IJCV)* (2009).

[173] Kramer, Michael, Dutkowski, Janusz, Yu, Michael, Bafna, Vineet, and Ideker, Trey. Inferring gene ontologies from pairwise similarity data. *Bioinformatics* (2014).

[174] Kranen, Philipp, Assent, Ira, Baldauf, Corinna, and Seidl, Thomas. The clustree: indexing micro-clusters for anytime stream mining. *Knowledge and Information systems* (2011).

[175] Krishnamurthy, Akshay, Balakrishnan, Sivaraman, Xu, Min, and Singh, Aarti. Efficient active algorithms for hierarchical clustering. *International Conference on Machine Learning (ICML)* (2012).

[176] Kulis, Brian, and Jordan, Michael I. Revisiting k-means: New algorithms via bayesian nonparametrics. *International Conference on Machine Learning (ICML)* (2012).

[177] Kushagra, Shrinu, Samadi, Samira, and Ben-David, Shai. Finding meaningful cluster structure amidst background noise. *Algorithmic Learning Theory (ALT)* (2016).

[178] Larson, Jeffrey S, Bradlow, Eric T., and Fader, Peter S. An exploratory look at supermarket shopping paths. *International Journal of Research in Marketing* (2005).

[179] Law, Marc Teva, Urtasun, Raquel, and Zemel, Richard S. Deep spectral clustering learning. *International Conference on Machine Learning (ICML)* (2017).

[180] Lee, Heeyoung, Recasens, Marta, Chang, Angel, Surdeanu, Mihai, and Jurafsky, Dan. Joint entity and event coreference resolution across documents. *Empirical Methods in Natural Language Processing (EMNLP)* (2012).

[181] Lee, Juho, and Choi, Seungjin. Bayesian hierarchical clustering with exponential family: Small-variance asymptotics and reducibility. *Artificial Intelligence and Statistics (AISTATS)* (2015).

[182] Lee, Kenton, He, Luheng, Lewis, Mike, and Zettlemoyer, Luke. End-to-end neural coreference resolution. *Empirical Methods in Natural Language Processing (EMNLP)* (2017).

[183] Levin, Michael, Krawczyk, Stefan, Bethard, Steven, and Jurafsky, Dan. Citation-based bootstrapping for large-scale author disambiguation. *Journal of the American Society for Information Science and Technology* (2012).

[184] Li, Bing, Wang, Wei, Sun, Yifang, Zhang, Linhan, Ali, Muhammad Asif, and Wang, Yi. Grapher: Token-centric entity resolution with graph convolutional neural networks. *AAAI Conference on Artificial Intelligence (AAAI)* (2020).

[185] Liberty, Edo, Sriharsha, Ram, and Sviridenko, Maxim. An algorithm for online k-means clustering. *Algorithm Engineering and Experiments (ALENEX)* (2016).

[186] Liu, Jialu, and Han, Jiawei. Spectral clustering. *Data Clustering* (2018).

[187] Liu, Jinze, Zhang, Qi, Wang, Wei, McMillan, Leonard, and Prins, Jan. Clustering pair-wise dissimilarity data into partially ordered sets. *Conference on Knowledge Discovery and Data Mining (KDD)* (2006).

[188] Liu, Jinze, Zhang, Qi, Wang, Wei, McMillan, Leonard, and Prins, Jan. Poclustering: Lossless clustering of dissimilarity data. *International Conference on Data Mining (ICDM)* (2007).

[189] Liu, Yang, Liu, Jing, Li, Zechao, Tang, Jinhui, and Lu, Hanqing. Weakly-supervised dual clustering for image semantic segmentation. *Computer Vision and Pattern Recognition (CVPR)* (2013).

[190] Liu, Yinhan, Ott, Myle, Goyal, Naman, Du, Jingfei, Joshi, Mandar, Chen, Danqi, Levy, Omer, Lewis, Mike, Zettlemoyer, Luke, and Stoyanov, Veselin. Roberta: A robustly optimized bert pretraining approach. *arXiv Pre-print* (2019).

[191] Lloyd, Stuart. Least squares quantization in pcm. *IEEE transactions on information theory* (1982).

[192] Logan IV, Robert L., McCallum, Andrew, Singh, Sameer, and Bikel, Daniel M. Benchmarking scalable methods for streaming cross document entity coreference. *Association for Computational Linguistics and International Joint Conference on Natural Language Processing (ACL-IJCNLP)* (2021).

[193] Lopez-Paz, David, and Ranzato, Marc'Aurelio. Gradient episodic memory for continual learning. *Advances in Neural Information Processing Systems (NeurIPS)* (2017).

[194] Low, Jia Shun, Ghafoori, Zahra, and Leckie, Christopher. Online k-means clustering with lightweight coresets. *Australasian Joint Conference on Artificial Intelligence* (2019).

[195] MacQueen, James, et al. Some methods for classification and analysis of multivariate observations. *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability* (1967).

[196] Mahajan, Meena, Nimbhorkar, Prajakta, and Varadarajan, Kasturi. The planar k-means problem is np-hard. *International Workshop on Algorithms and Computation* (2009).

[197] Mairal, Julien, Bach, Francis, Ponce, Jean, and Sapiro, Guillermo. Online dictionary learning for sparse coding. *International Conference on Machine Learning (ICML)* (2009).

[198] Malkov, Yury, Ponomarenko, Alexander, Logvinov, Andrey, and Krylov, Vladimir. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems* (2014).

[199] Manning, Christopher D., Raghavan, Prabhakar, and Schütze, Hinrich. Introduction to information retrieval, 2008.

[200] McCloskey, Michael, and Cohen, Neal J. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of learning and motivation* (1989).

[201] Mémoli, Facundo, and Okutan, Osman Berat. Reeb posets and tree approximations. *Discrete Mathematics* (2020).

[202] Menestrina, David, Whang, Steven Euijong, and Garcia-Molina, Hector. Evaluating entity resolution results. *VLDB* (2010).

[203] Menon, Aditya Krishna, Rajagopalan, Anand, Sumengen, Baris, Citovsky, Gui, Cao, Qin, and Kumar, Sanjiv. Online hierarchical clustering approximations. *arXiv Pre-print* (2019).

[204] Merity, Stephen, Xiong, Caiming, Bradbury, James, and Socher, Richard. Pointer sentinel mixture models. *arXiv Pre-print* (2016).

[205] Mikolov, Tomas, Grave, Edouard, Bojanowski, Piotr, Puhrsch, Christian, and Joulin, Armand. Advances in pre-training distributed word representations. *Conference on Language Resources and Evaluation (LREC 2018)* (2018).

[206] Miller, George A. Wordnet: a lexical database for english. *Communications of the ACM* (1995).

[207] Miller, George A. Wordnet: An electronic lexical database, 1998.

[208] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature* (2015).

[209] Monath, Nicholas, Dubey, Avinava, Guruganesh, Guru, Zaheer, Manzil, Ahmed, Amr, McCallum, Andrew, Mergen, Gokhan, Najork, Marc, Terzihan, Mert, Tjanaka, Bryon, et al. Scalable hierarchical agglomerative clustering. *Conference on Knowledge Discovery and Data Mining (KDD)* (2021).

[210] Monath, Nicholas, Kobren, Ari, Krishnamurthy, Akshay, Glass, Michael R, and McCallum, Andrew. Scalable hierarchical clustering with tree grafting. *Conference on Knowledge Discovery and Data Mining (KDD)* (2019).

[211] Monath, Nicholas, Zaheer, Manzil, Silva, Daniel, McCallum, Andrew, and Ahmed, Amr. Gradient-based hierarchical clustering using continuous representations of trees in hyperbolic space. *Conference on Knowledge Discovery and Data Mining (KDD)* (2019).

[212] Moretti, Antonio Khalil, Zhang, Liyi, Naesseth, Christian A, Venner, Hadiah, Blei, David, and Pe'er, Itsik. Variational combinatorial sequential monte carlo methods for Bayesian phylogenetic inference. *Conference on Uncertainty in Artificial Intelligence (UAI)* (2021).

[213] Moseley, Benjamin, Lu, Kefu, Lattanzi, Silvio, and Lavastida, Thomas. A framework for parallelizing hierarchical clustering methods. *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)* (2019).

[214] Moseley, Benjamin, and Wang, Joshua. Approximation bounds for hierarchical clustering: Average linkage, bisecting k-means, and local search. *Advances in Neural Information Processing Systems (NeurIPS)* (2017).

[215] Moshkovitz, Michal. Unexpected effects of online no-substitution k-means clustering. *Conference on Algorithmic Learning Theory (ALT)* (2019).

[216] Müllner, Daniel. Modern hierarchical, agglomerative clustering algorithms. *arXiv Pre-print* (2011).

[217] Murtagh, F. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal* (1983).

[218] Murtagh, Fionn. Complexities of hierarchic clustering algorithms: State of the art. *Computational Statistics Quarterly* (1984).

[219] Naumov, Stanislav, Yaroslavtsev, Grigory, and Avdiukhin, Dmitrii. Objective-based hierarchical clustering of deep embedding vectors. *AAAI Conference on Artificial Intelligence (AAAI)* (2021).

[220] Neal, Radford M. Markov chain sampling methods for Dirichlet process mixture models. *Journal of Computational and Graphical Statistics* (2000).

[221] Neal, Radford M. Density modeling and clustering using Dirichlet diffusion trees. *Bayesian Statistics* (2003).

[222] Ng, Andrew Y, Jordan, Michael I, and Weiss, Yair. On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems (NeurIPS)* (2002).

[223] Nguyen, Thuy-Diem, Schmidt, Bertil, and Kwoh, Chee-Keong. Sparsehc: a memory-efficient online hierarchical clustering algorithm. *Procedia Computer Science* (2014).

[224] O'Callaghan, Liadan, Mishra, Nina, Meyerson, Adam, Guha, Sudipto, and Motwani, Rajeev. Streaming-data algorithms for high-quality clustering. *International Conference on Data Engineering (ICDE)* (2002).

[225] Olson, Clark F. Parallel algorithms for hierarchical clustering. *Parallel computing* (1995).

[226] Pagliardini, Matteo, Gupta, Prakhar, and Jaggi, Martin. Unsupervised learning of sentence embeddings using compositional n-gram features. *NAACL* (2018).

[227] Paisley, John, Wang, Chong, Blei, David M, and Jordan, Michael I. Nested hierarchical Dirichlet processes. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2014).

[228] Pan, Xinghao, Gonzalez, Joseph E, Jegelka, Stefanie, Broderick, Tamara, and Jordan, Michael I. Optimistic concurrency control for distributed unsupervised learning. *Advances in Neural Information Processing Systems (NeurIPS)* (2013).

[229] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research (JMLR)* (2011).

[230] Pelleg, Dan, Moore, Andrew W, et al. X-means: Extending k-means with efficient estimation of the number of clusters. *International Conference on Machine Learning (ICML)* (2000).

[231] Peng, Min, Zhu, Jiahui, Wang, Hua, Li, Xuhui, Zhang, Yanchun, Zhang, Xiuzhen, and Tian, Gang. Mining event-oriented topics in microblog stream with unsupervised multi-view hierarchical embedding. *Conference on Knowledge Discovery and Data Mining (KDD)* (2018).

[232] Pradhan, Sameer, Luo, Xiaoqiang, Recasens, Marta, Hovy, Eduard, Ng, Vincent, and Strube, Michael. Scoring coreference partitions of predicted mentions: A reference implementation. *Association for Computational Linguistics (ACL)* (2014).

[233] Preska Steinberg, Asher, Lin, Mingzhi, and Kussell, Edo. Core genes can have higher recombination rates than accessory genes within global microbial populations. *bioRxiv* (2021).

[234] Pritzel, Alexander, Uria, Benigno, Srinivasan, Sriram, Puigdomenech, Adria, Vinyals, Oriol, Hassabis, Demis, Wierstra, Daan, and Blundell, Charles. Neural episodic control. *International Conference on Machine Learning (ICML)* (2017).

[235] Rajagopalan, Anand, Vitale, Fabio, Vainstein, Danny, Citovsky, Gui, Procopiuc, Cecilia M., and Gentile, Claudio. Hierarchical clustering of data streams: Scalable algorithms and approximation guarantees. *International Conference on Machine Learning (ICML)* (2021).

[236] Rand, William M. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association (JASA)* (1971).

[237] Rao, Delip, McNamee, Paul, and Dredze, Mark. Streaming cross document entity coreference resolution. *Conference on Computational Linguistics (COLING)* (2010).

[238] Reif, John H., and Tate, Stephen R. Dynamic parallel tree contraction. *Symposium on Parallel algorithms and architectures (SPAA)* (1997).

[239] Remm, Maido, Storm, Christian EV, and Sonnhammer, Erik LL. Automatic clustering of orthologs and in-paralogs from pairwise species comparisons. *Journal of molecular biology* (2001).

[240] Roditty, Liam, and Zwick, Uri. Improved dynamic reachability algorithms for directed graphs. *Foundations of Computer Science (FOCS)* (2002).

[241] Rohlf, F. James. Hierarchical clustering using the minimum spanning tree. *Comput. Journal* (1973).

[242] Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)* (2015).

[243] Santos, Joelson, Syed, Talat, Naldi, Murilo Coelho, Campello, Ricardo JGB, and Sander, Jörg. Hierarchical density-based clustering using mapreduce. *IEEE Transactions on Big Data* (2019).

[244] Schmidt, Melanie, Schwiegelshohn, Chris, and Sohler, Christian. Fair coresets and streaming algorithms for fair k-means clustering. *arXiv Pre-print* (2019).

[245] Schwartz, Gregory W, Zhou, Yeqiao, Petrovic, Jelena, Fasolino, Maria, Xu, Lanwei, Shaffer, Sydney M, Pear, Warren S, Vahedi, Golnaz, and Faryabi, Robert B. TooManyCells identifies and visualizes relationships of single-cell clades. *Nature Methods* (2020).

[246] Sculley, D. Web-scale k-means clustering. *The Web Conference (WWW)* (2010).

[247] Shaham, Uri, Stanton, Kelly P., Li, Haochao, Nadler, Boaz, Basri, Ronen, and Kluger, Yuval. Spectralnet: Spectral clustering using deep neural networks. *arXiv Pre-print* (2018).

[248] Shamir, Ohad, and Tishby, Naftali. Spectral clustering on a budget. *Artificial Intelligence and Statistics (AISTATS)* (2011).

[249] Shang, Fanhua, Jiao, LC, Shi, Jiarong, Wang, Fei, and Gong, Maoguo. Fast affinity propagation clustering: A multilevel approach. *Pattern recognition* (2012).

[250] Shi, Jianbo, and Malik, Jitendra. Normalized cuts and image segmentation. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2000).

[251] Shiloach, Yossi, and Vishkin, Uzi. An o (log n) parallel connectivity algorithm. Tech. rep., Computer Science Department, Technion, 1980.

[252] Shiokawa, Hiroaki. Scalable affinity propagation for massive datasets. *AAAI Conference on Artificial Intelligence (AAAI)* (2021).

[253] Sibson, Robin. Slink: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal* (1973).

[254] Silverman, Bernard W. Density estimation for statistics and data analysis, 1986.

[255] Singh, Sameer, Subramanya, Amarnag, Pereira, Fernando, and McCallum, Andrew. Large-scale cross-document coreference using distributed inference and hierarchical models. *Association for Computational Linguistics: Human Language Technologies (ACL-HLT)* (2011).

[256] Sleator, Daniel Dominic, and Tarjan, Robert E. A data structure for dynamic trees. *Journal of Computer and System Sciences (JCSS)* (1981).

[257] Sørlie, Therese, Perou, Charles M, Tibshirani, Robert, Aas, Turid, Geisler, Stephanie, Johnsen, Hilde, Hastie, Trevor, Eisen, Michael B, Van De Rijn, Matt, Jeffrey, Stefanie S, et al. Gene expression patterns of breast carcinomas distinguish tumor subclasses with clinical implications. *Proceedings of the National Academy of Sciences* (2001).

[258] Steorts, Rebecca C., Barnes, Matt, and Neiswanger, Willie. Performance bounds for graphical record linkage. *Artificial Intelligence and Statistics (AISTATS)* (2017).

[259] Steorts, Rebecca C., Hall, Rob, and Fienberg, Stephen E. SMERED: A Bayesian approach to graphical record linkage and de-duplication. *Artificial Intelligence and Statistics (AISTATS)* (2014).

[260] Tarjan, Robert E., and Werneck, Renato F. Self-adjusting top trees. *Symposium on Discrete Algorithms (SODA)* (2005).

[261] Tarjan, Robert E., and Werneck, Renato F. Dynamic trees in practice. *ACM Journal of Experimental Algorithmics (JEA)* (2010).

[262] Teh, Yee Whye, Daume III, Hal, and Roy, Daniel M. Bayesian agglomerative clustering with coalescents. *Advances in Neural Information Processing Systems (NeurIPS)* (2008).

[263] Telgarsky, Matus, and Dasgupta, Sanjoy. Agglomerative bregman clustering. *International Conference on Machine Learning (ICML)* (2012).

[264] Telgarsky, Matus, and Vattani, Andrea. Hartigan's method: k-means clustering without voronoi. *Artificial Intelligence and Statistics (AISTATS)* (2010).

[265] Tian, Fei, Gao, Bin, Cui, Qing, Chen, Enhong, and Liu, Tie-Yan. Learning deep representations for graph clustering. *AAAI Conference on Artificial Intelligence (AAAI)* (2014).

[266] Tseng, Tom, Dhulipala, Laxman, and Blelloch, Guy E. Batch-parallel euler tour trees. *Algorithm Engineering and Experiments (ALENEX)* (2019).

[267] Vamosi, Steven Michael, Heard, Stephen B., Vamosi, Jana C., and Webb, Campbell O. Emerging patterns in the comparative analysis of phylogenetic community structure. *Molecular Ecology* (2009).

[268] van de Ven, Gido M, and Tolias, Andreas S. Generative replay with feedback connections as a general strategy for continual learning. *Advances in Neural Information Processing Systems (NeurIPS)* (2017).

[269] Vandic, Damir, Frasincar, Flavius, Kaymak, Uzay, and Riezebos, Mark. Scalable entity resolution for web product descriptions. *Information Fusion* (2020).

[270] Vassilvitskii, Sergei, and Arthur, David. k-means++: The advantages of careful seeding. *Symposium on Discrete Algorithms (SODA)* (2006).

[271] Vazirani, Vijay V. Approximation algorithms, 2013.

[272] Vendrov, Ivan, Kiros, Ryan, Fidler, Sanja, and Urtasun, Raquel. Order-embeddings of images and language. *arXiv Pre-print* (2015).

[273] Vikram, Sharad, Hoffman, Matthew D., and Johnson, Matthew J. The loracs prior for vaes: Letting the trees speak for the data. *Artificial Intelligence and Statistics (AISTATS)* (2019).

[274] Vilnis, Luke, Li, Xiang, Murty, Shikhar, and McCallum, Andrew. Probabilistic embedding of knowledge graphs with box lattice measures. *Association for Computational Linguistics (ACL)* (2018).

[275] Vitale, Fabio, Rajagopalan, Anand, and Gentile, Claudio. Flattening a hierarchical clustering through active learning. *Advances in Neural Information Processing Systems (NeurIPS)* (2019).

[276] Voges, Kevin E., Pope, Nigel K. L., and Brown, Mark. Cluster analysis of marketing data examining on-line shopping orientation: a comparison of k-means and rough clustering approaches, 2002.

[277] Wang, Dingkang, and Wang, Yusu. A new cost function for hierarchical cluster trees. *arXiv Pre-print* (2018).

[278] Wang, Kaijun, Zhang, Junying, Li, Dan, Zhang, Xinna, and Guo, Tao. Adaptive affinity propagation clustering. *arXiv Pre-print* (2008).

[279] Wang, Liangliang, Bouchard-Côté, Alexandre, and Doucet, A. Bayesian phylogenetic inference using a combinatorial sequential monte carlo method. *Journal of the American Statistical Association (JASA)* (2015).

[280] Wang, Xiting, Liu, Shixia, Song, Yangqiu, and Guo, Baining. Mining evolutionary multi-branch trees from text streams. *Conference on Knowledge Discovery and Data Mining (KDD)* (2013).

[281] Ward, Joe H. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association (JASA)* (1963).

[282] Watts, Duncan J, and Strogatz, Steven H. Collective dynamics of 'small-world'networks. *Nature* (1998).

[283] Wick, Michael, Singh, Sameer, and McCallum, Andrew. A discriminative hierarchical model for fast coreference at large scale. *Association for Computational Linguistics (ACL)* (2012).

[284] Widyantoro, Dwi H, Ioerger, Thomas R, and Yen, John. An incremental approach to building a cluster hierarchy. *International Conference on Data Mining (ICDM)* (2002).

[285] Williamson, Sinead, Dubey, Avinava, and Xing, Eric. Parallel Markov chain Monte Carlo for nonparametric mixture models. *International Conference on Machine Learning (ICML)* (2013).

[286] Wiseman, Sam, Rush, Alexander M, and Shieber, Stuart M. Learning global features for coreference resolution. *NAACL* (2016).

[287] Xie, Junyuan, Girshick, Ross B., and Farhadi, Ali. Unsupervised deep embedding for clustering analysis. *arXiv Pre-print* (2016).

[288] Yadav, Nishant, Kobren, Ari, Monath, Nicholas, and McCallum, Andrew. Supervised hierarchical clustering with exponential linkage. *International Conference on Machine Learning (ICML)* (2019).

[289] Yan, Donghui, Huang, Ling, and Jordan, Michael I. Fast approximate spectral clustering. *Conference on Knowledge Discovery and Data Mining (KDD)* (2009).

[290] Yang, Jianwei, Parikh, Devi, and Batra, Dhruv. Joint unsupervised learning of deep representations and image clusters. *Computer Vision and Pattern Recognition (CVPR)* (2016).

[291] Yaroslavtsev, Grigory, and Vadapalli, Adithya. Massively parallel algorithms and hardness for single-linkage clustering under ℓ_p distances. *International Conference on Machine Learning (ICML)* (2018).

[292] Yen, Ian En-Hsu, Malioutov, Dmitry, and Kumar, Abhishek. Scalable exemplar clustering and facility location via augmented block coordinate descent with column generation. *Artificial Intelligence and Statistics (AISTATS)* (2016).

[293] Yu, Jie, Tao, Chao, Xu, Lingyu, Wu, Haiqiao, and Liu, Fangfang. Construction of hierarchical cognitive academic map. *IEEE Access* (2017).

[294] Zaheer, Manzil, Ahmed, Amr, and Smola, Alexander J. Latent LSTM allocation: Joint clustering and non-linear dynamic modeling of sequence data. *International Conference on Machine Learning (ICML)* (2017).

[295] Zaheer, Manzil, Kottur, Satwik, Ahmed, Amr, Moura, José, and Smola, Alex. Canopy—fast sampling with cover trees. *International Conference on Machine Learning (ICML)* (2017).

[296] Zaheer, Manzil, Wick, Michael, Tristan, Jean-Baptiste, Smola, Alex, and Steele, Guy. Exponential stochastic cellular automata for massively parallel inference. *Artificial Intelligence and Statistics (AISTATS)* (2016).

[297] Zhan, Xiaohang, Xie, Jiahao, Liu, Ziwei, Ong, Yew Soon, and Loy, Chen Change. Online deep clustering for unsupervised representation learning. *Computer Vision and Pattern Recognition (CVPR)* (2020).

[298] Zhang, Aonan, and Paisley, John. Markov mixed membership models. *International Conference on Machine Learning (ICML)* (2015).

[299] Zhang, Luming, Song, Mingli, Zhao, Qi, Liu, Xiao, Bu, Jiajun, and Chen, Chun. Probabilistic graphlet transfer for photo cropping. *IEEE Transactions on Image Processing* (2012).

[300] Zhang, Tian, Ramakrishnan, Raghu, and Livny, Miron. Birch: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery* (1997).

[301] Zhang, Yongzhe, Azad, Ariful, and Hu, Zhenjiang. Fastsv: A distributed-memory connected component algorithm with fast convergence. *Conference on Parallel Processing for Scientific Computing* (2020).

[302] Zhang, Yuchen, Ahmed, Amr, Josifovski, Vanja, and Smola, Alexander. Taxonomy discovery for personalized recommendation. *Conference on Web Search and Data Mining (WSDM)* (2014).

[303] Zhang, Yutao, Zhang, Fanjin, Yao, Peiran, and Tang, Jie. Name disambiguation in AMiner: Clustering, maintenance, and human in the loop. *Conference on Knowledge Discovery and Data Mining (KDD)* (2018).

[304] Zhao, Jian, Chevalier, Fanny, Collins, Christopher, and Balakrishnan, Ravin. Facilitating discourse analysis with interactive visualization. *IEEE Transactions on Visualization and Computer Graphics* (2012).

[305] Zhao, Jinyu, Hao, Yi, and Rashtchian, Cyrus. Unsupervised embedding of hierarchical structure in euclidean space. *arXiv Pre-print* (2020).

[306] Zhao, Ying, and Karypis, George. Evaluation of hierarchical clustering algorithms for document datasets. *Conference on Information and Knowledge Management (CIKM)* (2002).

[307] Zhong, Shi. Efficient online spherical k-means clustering. *International Joint Conference on Neural Networks (IJCNN)* (2005).