Thesis for The Degree of Doctor of Philosophy

Clustering in the Big Data Era: methods for efficient approximation, distribution, and parallelization

Amir Keramatian

Division of Networks and Systems
Department of Computer Science & Engineering
Chalmers University of Technology | University of Gothenburg
Gothenburg, Sweden, 2022

Clustering in the Big Data Era: methods for efficient approximation, distribution, and parallelization

Amir Keramatian

ISBN 978-91-7905-650-6
Doktorsavhandlingar vid Chalmers tekniska högskola, Ny serie nr 5116.
ISSN 0346-718X

Technical Report No 217D
Department of Computer Science & Engineering
Division of Networks and Systems
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Digitaltryck,
Gothenburg, Sweden 2022.

Abstract

Data clustering is an unsupervised machine learning task whose objective is to group together similar items. As a versatile data mining tool, data clustering has numerous applications, such as object detection and localization using data from 3D laser-based sensors, finding popular routes using geolocation data, and finding similar patterns of electricity consumption using smart meters.

The datasets in modern IoT-based applications are getting more and more challenging for conventional clustering schemes. *Big Data* is a term used to loosely describe hard-to-manage datasets. Particularly, large numbers of data points, high rates of data production, large numbers of dimensions, high skewness, and distributed data sources are aspects that challenge the classical data processing schemes, including clustering methods.

This thesis contributes to efficient big data clustering for distributed and parallel computing architectures, representative of the processing environments in edge-cloud computing continuum. The thesis also proposes approximation techniques to cope with certain challenging aspects of big data.

Regarding *distributed clustering*, the thesis proposes MAD-C, abbreviating Multi-stage Approximate Distributed Cluster-Combining. MAD-C leverages an approximation-based data synopsis that drastically lowers the required communication bandwidth among the distributed nodes and achieves multiplicative savings in computation time, compared to a baseline that centrally gathers and clusters the data. The thesis shows MAD-C can be used to detect and localize objects using data from distributed 3D laser-based sensors with high accuracy. Furthermore, the work in the thesis shows how to utilize MAD-C to efficiently detect the objects within a restricted area for geofencing purposes.

Regarding *parallel clustering*, the thesis proposes a family of algorithms called PARMA-CC, abbreviating Parallel Multistage Approximate Cluster Combining. Using approximation-based data synopsis, PARMA-CC algorithms achieve scalability on multi-core systems by facilitating parallel execution of threads with limited dependencies which get resolved using fine-grained synchronization techniques. To further enhance the efficiency, PARMA-CC algorithms can be configured with respect to different data properties. Analytical and empirical evaluations show PARMA-CC algorithms achieve significantly higher scalability than the state-of-the-art methods while preserving a high accuracy.

On *parallel high dimensional clustering*, the thesis proposes IP.LSH.DBSCAN, abbreviating Integrated Parallel Density-Based Clustering through Locality-Sensitive Hashing (LSH). IP.LSH.DBSCAN fuses the process of creating an LSH index into the process of data clustering, and it takes advantage of data parallelization and fine-grained synchronization. Analytical and empirical evaluations show IP.LSH.DBSCAN facilitates parallel density-based clustering of massive datasets using desired distance measures resulting in several orders of magnitude lower latency than state-of-the-art for high dimensional data.

In essence, the thesis proposes methods and algorithmic implementations targeting the problem of big data clustering and applications using distributed and parallel processing. The proposed methods (available as open source software) are extensible and can be used in combination with other methods.

Keywords: Clustering, Applied ML, Approximation-based synopsis, Distributed and Parallel Processing

Acknowledgments

I would like to start by expressing my deepest gratitude towards my supervisors Marina Papatriantafilou, Vincenzo Gulisano, and Philippas Tsigas for their precious guidance, support, and patience. I would also like to express my heartfelt appreciation towards Agneta Nilsson and Tomas Olovsson for helping me through my PhD journey.

I am honored to have Assoc. Prof. Ioannis Chatzigiannakis as the faculty opponent for my PhD defence. I would also like to thank members of the grading committee: Prof. Paola Flocchini, Prof. Ralf Klasing, Prof. Vladimir Vlassov, and Prof. Ulf Assarsson. I also would like to express my gratitude towards my examiner Prof. Jan Jonsson and also all the follow-up groups during my PhD.

I am grateful for all the support and assistance from the department's administration staff Eva Axelsson, Rebecca Cyren, Jenny Lind, Monica Månhammar, Michael Morin, Lars Noren, Clara Oders, and Marianne Pleen-Schreiber.

I take this opportunity to thank past and present colleagues in the division. Many thanks to Adones, Ahmed, Ali, Aljoscha, Aras, Bapi, Bastian, Bei, Beshr, Boel, Carlo, Charalampos, Christos, Dimitris, Elad, Fazeleh, Francisco, Georgia, Hannah, Yiannis, Iosif, Ismail, Ivan, Joris, Karl, Katerina, Magnus, Nasser, Olaf, Oliver, Paul, Romaric, Thomas, Valentin T., Valentin P., and Wissam.

I would not have been able to come this far if it were not for my parents' endless love and support. I am grateful also for my friends Alireza, Hamid, Mozhgan, and Sharon. Last but not least, to my wonderful Karolina Maria, I feel blessed because of you; thank you and bardzo Cie kocham!

Funding. I would like to acknowledge the financial support by the Swedish Foundation for Strategic Research, project Factories in the Cloud (FiC), with grant number GMT14-0032.

Amir Keramatian
Göteborg, April 2022

List of Publications

Appended publications

This thesis is based on the following publications:

[A] **Amir Keramatian**, Vincenzo Gulisano, Marina Papatriantafilou, Philippas Tsigas, "MAD-C: Multi-stage Approximate Distributed Cluster-combining for obstacle detection and localization", in *the Journal of Parallel and Distributed Computing (JPDC)*, vol. 147, pp. 248-267, Elsevier, 2021.

The above is an extended and elaborated version of the work that previously appeared in:

Amir Keramatian, Vincenzo Gulisano, Marina Papatriantafilou, Philippas Tsigas, Yiannis Nikolakopoulos, "MAD-C: Multi-stage Approximate Distributed Cluster-combining for obstacle detection and localization", in *Euro-Par 2018: Parallel Processing Workshops: Euro-Par 2018 International Workshops, Turin, Italy, August 27-28, 2018, Revised Selected Papers*, vol. 11339, pp. 312-324. Springer, 2018.

[B] **Amir Keramatian**, Vincenzo Gulisano, Marina Papatriantafilou, Philippas Tsigas, "PARMA-CC: A Family of Parallel Multiphase Approximate Cluster Combining Algorithms", *the Journal of Parallel and Distributed Computing (JPDC)*, Under Review After Minor Revision, Elsevier, 2022.

The above is an extended and elaborated version of the work that previously appeared in:

Amir Keramatian, Vincenzo Gulisano, Marina Papatriantafilou, Philippas Tsigas, "PARMA-CC: Parallel Multiphase Approximate Cluster Combining", in *the Proceedings of the 21st International Conference on Distributed Computing and Networking (ICDCN)*, pp. 20:1-20:10, ACM, 2020.

[C] **Amir Keramatian**, Vincenzo Gulisano, Marina Papatriantafilou, Philippas Tsigas, "IP.LSH.DBSCAN: Integrated Parallel Density-Based Clustering through Locality-Sensitive Hashing", Under Review.

Research Contributions

I contributed to all the listed articles as the lead designer, author, and also the sole implementer.

Contents

Chapter 1

Overview

1.1 Introduction

Machine learning is one of the fastest growing technical fields which has been extensively applied throughout science, technology, and commerce [1]. Data clustering is an *unsupervised* machine learning task with the objective of discovering groups of similar samples within a given untagged dataset. Data clustering facilitates discovering similarities and differences among patterns and inferring useful conclusions [2]. The concept of data clustering naturally appears in many fields such as biology, zoology, sociology, archaeology, geology and engineering. Because of its unsupervised nature and versatility, data clustering is a fundamental data mining problem arising in many applications [3]. For instance, data clustering has been leveraged for object detection [4–6], route analysis [7–9], digital image processing [10, 11], text processing [12], and audio clustering [13]. Despite all the advancements in data clustering, *big data* challenges even the most established methods in the field, where big data is a term used to refer to hard-to-manage data. As the challenging aspects of big data keep intensifying in modern datasets, there is a large room for improved clustering methods that can cope with various aspects of big data.

For improving clustering methods with respect to the challenges imposed by big data, the advantages and limitations of the computational infrastructure has to be well understood. The following introduces the elements of a *multi-tier* computing infrastructure.

Cloud Computing. It is a common practice to transfer data to the *cloud*, where storage, computation, analysis, and decision making take place [14]. As the typical infrastructure in cloud provides abundant resources (e.g., power-full high-end servers), *cloud computing* can be used to solve various computational problems. Cloud computing has several advantages. Firstly, computational speedup can be achieved by properly leveraging the resources of high-end servers. Secondly, the cloud services typically enable the users to pay for the services in a pay-as-you-go manner, making it more economical than having on-premise equipment. Thirdly, the infrastructure is maintained by the cloud service providers, sparing the end users from technical managements. These advantages show the importance of cloud computing for many problems in IoT-

based systems. On the other hand, cloud computing has some disadvantages as well. Considering the distance between the devices and cloud service providers, the services might suffer from high latency. In addition, transmitting the data from all the devices in the network requires high bandwidth. Furthermore, even if a high bandwidth data communication channel is utilized, concurrent transmission of large volume of data does not scale with increasing number of transmitters as the communication infrastructure might experience *jitter* [15] and finally *collapse* [16].

Fog and Edge Computing. Edge computing refers to the computation, analysis, storage, and decision making that takes place at *the edge of the network*, i.e., the devices (like mobile phones) directly connected to the sensors or an IoT gateway (e.g., a device that aggregates and transmits sensor data) closely located to the sensors. *Fog computing* refers to leveraging the computational and storage capacities within a local network of systems (e.g., a LAN) to carry out the computation that would, otherwise, require the resources of a cloud infrastructure. Compared to cloud computing, fog and edge computing facilitate processing data closer to where it is produced [14]. As a result, adapting IoT-based systems to fog/edge computing can provide solutions with lower latency and higher security, as the data remains within the local network. Nevertheless, fog/edge computing can impose additional challenging as fog/edge devices are typically resource-constrained in terms of computational power. Furthermore, such devices can be connected via limited bandwidth media, e.g., wireless networks, which makes transmitting large volumes of data expensive.

The remainder of this chapter is organized as follows: § 1.2 introduces the research scope of the thesis. § 1.3 lays out the required preliminaries of the overview chapter. The research challenges within the scope of the thesis are presented in § 1.4, while § 1.5 outlines the contributions of thesis towards the research challenges. § 1.6 elaborates on how the thesis advances the state-of-the-art. Finally, § 1.7 concludes the chapter and provides suggestions for future research.

1.2 The Scope of the Thesis

This thesis targets the problem of efficient big data clustering using distributed and parallel computing. The subsequent subsections elaborate on the key aspects of the scope of the thesis.

1.2.1 Challenging Aspects of Data

The following elaborates on the challenging aspects of big data that this thesis considers:

- *Large Volumes*: Processing too large volumes of data with any data processing method whose computational complexity is higher than almost-linear requires excessive amounts of time which can be detrimental to time-sensitive problems [17, 18].

- *High Dimensionality*: Curse of dimensionality [19] is a term used to describe the collection of challenges imposed by high data dimensionality. For instance, the volume of the space in which data resides increases exponentially with increasing number of dimensions. The latter poses significant challenges to classical data processing techniques [20].

- *Diverse Properties*: Depending on the application and how data is gathered, the data can exhibit a wide range of properties which can play an adversary against most classical processing techniques. For example, highly skewed data distributions cause spatial indexing data structures suffer from query performance degradation [21–23].

- *Distributed Data Sources*: Processing data whose parts get distributedly gathered (e.g., by distributed sensors) imposes communication and computation challenges [17]. For example, a network of Automated Guided Vehicle (AGV), for the purpose of navigation and avoiding obstacles, might be equipped with high-rate laser-based sensors. As the aforementioned AGVs gather complementing data, it is beneficial to utilize all the data parts in the processing scheme. However, architecting an efficient solution with respect to the communication and computation latency is challenging.

1.2.2 Methods for Big Data Processing and the Associated Challenges

To address the challenges of processing big data, the methods proposed in this thesis use the insights in P. Gibbons' keynote [24]: (i) *scaling down* the amount of data to be processed, (ii) utilizing parallel processing and efficient use of shared memory to *scale up* the computing on a node, and (iii) utilizing distributed nodes (e.g., in fog/edge) to *scale out* the computing to different nodes. The following expands on the potential opportunities and challenges of aforementioned guidelines.

Scaling Down the Data

The first line of defence against big data is scaling down the large volume of data. Scaling down can be applied in a variety of forms [25], for instance, sampling [18, 26], dimensionality reduction [27], compression [28], and bucketization [29, 30].

Many applications do not require exact solutions; however, they need to make decisions as fast as possible [31]. For instance, regarding the problem of managing AGVs, a real-time solution that approximately detects the objects in the environment and avoids hitting them is more helpful than an exact solution that detects the objects with all the details but fails to finish within reasonable amount of time.

Approximation techniques can be employed to derive a data synopsis orders of magnitude smaller than the original data, leading to orders of magnitude speed-up in processing time. Nevertheless, there are challenges in designing and employing effective approximation approaches with regards to time, space, practicality, and accuracy factors [17]. For instance, considering the large volume of data, it is often important that the data synopsis be constructed

in only one pass over the original data. A data synopsis should also take into account other data-induced challenges such as high dimensionality, where applicable.

Scaling up the Computing in one Node

Modern computing platforms support concurrent execution, synchronization, and communication of many workers (e.g., threads, CPUs, etc), sharing the available resources of the system, for instance memory. Leveraging parallel processing and efficient use of the shared memory can lead to scaling up the computing on a single node. Maximizing such scalability is particularly important on powerful high-end servers (e.g., those typically employed in the cloud computing) as they provide abundant resources in terms of computational power, memory, and storage. For example, a high-end server can consist of several sockets, each socket accommodating several cores, and each core can possibly support hyper-threading [32].

However, regarding many problems in IoT-based systems, achieving and maintaining high scalability with increasing number of workers is challenging. In fact, increasing the number of workers might even increase computational latency. The following presents a discussion on two scalability models in order to shed light on the scalability challenges.

Scalability Models. Amdahl's law [33] states if a portion of a system can be improved but the other portion can not be improved, then the unimprovable portion becomes the dominating factor of the execution. Concretely, Amdahl's characterizes the maximum scalability of a system as $\frac{\mathtt{K}}{\mathtt{p}+(1-\mathtt{p})\mathtt{K}}$, where $\mathtt{p}$ denotes the improvable portion of the system and $\mathtt{K}$ is the number of workers. In the context of the thesis, the execution latency of a task is at least equal to the duration of its non-parallelizable parts, and further parallelization can not improve the latency. USL, or the universal scalability law [34], proposed by Neil Gunther, is another model to quantify the scalability. According to USL, there are two main obstacles on the way of achieving linear scalability with the number of workers. The first one is *contention*, and the second one is *crosstalk* [35]. Contention happens when the workers of a task require a shared resource but can not have it at the same time. Therefore, they have to queue up, which negatively affects the parallelization of the task. Crosstalk happens when the pairs of workers need to communicate in order to synchronize and share their states, which negatively affects the scalability. Based on contention and crosstalk, USL models the scalability of a system as $\frac{\mathtt{K}}{1+\delta(\mathtt{K}-1)+\kappa\mathtt{K}(\mathtt{K}-1)}$, where $\mathtt{K}$ is the number of workers, and δ is the contention degree coefficient, and κ is the crosstalk penalty coefficient. Note δ gets multiplied by $(\mathtt{K}-1)$ which shows contention grows linearly in the number of workers (i.e., as they queue up for a shared resource). On the other hand, κ gets multiplied by $\mathtt{K}(\mathtt{K}-1)$ reflecting the crosstalk between all pairs of workers in the system. Note that, in an ideal system, coefficients δ and κ are zero; therefore, the scalability of the system increases linearly with the number of workers. Nevertheless, with κ and δ being greater than zero, the growth of denominator gets higher than the numerator, and the speed-up will start to decrease after a large enough value of $\mathtt{K}$.

Based on Amdahl's law, to achieve high scalability, the challenge is to design computational tasks with minimum non-parallelizable parts. Furthermore, based on USL, data structures, algorithms, and work sharing schemes should be designed to minimize the contention for the shared resources and the crosstalk among the pairs of workers.

Scaling out the Computing to Distributed Nodes in Fog/Edge

Increasing the number of computing nodes is another approach to deal with big data [24, 36, 37]. To that end, an interconnected network of fog/edge devices (with attached sensors) can be leveraged to process the gathered sensor data. As the communication takes place in the local network of the devices, the communication latency is lower than the cloud computing setup, and the privacy of data is preserved within the boundaries of the local network.

Nevertheless, there are two challenges that need to be considered in designing an IoT-based system that scales out the computing. The first one concerns the fact that the fog/edge devices are weaker than the high-end servers, in the sense that they have limited computational and memory capacities. The second challenge concerns the limitations associated with the shared communication medium such as limited bandwidth and jitter.

1.3 Preliminaries

This section introduces highlights of background that is useful when reading the subsequent sections.

1.3.1 Examples of Data in IoT Applications

LIDAR (LIght Detection And Ranging) is a scanning method to generate a 3-D representation of a target by illuminating the target with pulsed light waves. The 3-D model is generated based on the time that light waves take to return. A LIDAR scanner typically leverages several laser beams (e.g., 8 to 128). Located on the spinning head of the sensor, each laser beam emits photons and reads back their reflection several times in a second. As the spinning head of the LIDAR scanner makes a full rotation, a 360 degree high resolution 3-D representation of the environment is attained [38]. The readings from a LIDAR sensor are named *point clouds* [39]. Point clouds generated by certain types of LIDAR sensors can contain hundreds of thousands of 3-D points, or even more. The point cloud from a full rotation of a LIDAR sensor's spinning head contains points corresponding to *scene objects* (e.g., pedestrians, vehicles, etc) in the environment in which the sensor is installed. The readings in a point cloud generated by such a sensor are sorted with respect to time and space. In general, the thesis uses the term *spatio-temporal locality* to indicate the extent to which the points in a dataset are sorted with respect to time and space.

In contrast to 3D point clouds, points in different datasets might contain much more number of features, i.e., dimensions. For instance, digital representation of images, audio, and video usually require much more number of features. Even a low-resolution 28×28-pixel image is digitally represented by a 784-feature vector [40].

1.3.2 Distance Measures

Given a domain $\mathtt{S}$, a function $\mathtt{Distance}$ from $\mathtt{S} \times \mathtt{S}$ to non-negative real numbers is called a *metric* [41] (or equivalently a distance measure) if it satisfies the following conditions for any $\mathtt{x}$, $\mathtt{y}$, $\mathtt{z} \in \mathtt{S}$:

[A] non-negativity: $\mathtt{Distance}(\mathtt{x}, \mathtt{y}) \geq 0$

[B] identity of indiscernibles: $\mathtt{Distance}(\mathtt{x}, \mathtt{y}) = 0 \iff \mathtt{x} = \mathtt{y}$

[C] symmetry: $\mathtt{Distance}(\mathtt{x}, \mathtt{y}) = \mathtt{Distance}(\mathtt{y}, \mathtt{x})$

[D] triangle inequality: $\mathtt{Distance}(\mathtt{x}, \mathtt{z}) \leq \mathtt{Distance}(\mathtt{x}, \mathtt{y}) + \mathtt{Distance}(\mathtt{y}, \mathtt{z})$

Among the metrics that satisfy above conditions are Euclidean distance, Manhattan distance, angular distance, edit distance, and hamming distance [18]. The following specifies Euclidean distance and angular distance since they are used in the thesis.

Euclidean Distance. For $\mathtt{x}$ and $\mathtt{y}$ in $\mathbb{R}^{\mathtt{d}}$, Euclidean distance measures the length of the line segment between $\mathtt{x}$ and $\mathtt{y}$, which is, following the Pythagorean formula, $\sqrt{(\mathtt{x}_1 - \mathtt{y}_1)^2 + \cdots + (\mathtt{x}_{\mathtt{d}} - \mathtt{y}_{\mathtt{d}})^2}$.

Angular Distance. Considering $\mathtt{x}$ and $\mathtt{y}$ as vectors in $\mathbb{R}^{\mathtt{d}}$, angular distance measures the angle that $\mathtt{x}$ and $\mathtt{y}$ make, which is $\arccos(\frac{\mathtt{x} \cdot \mathtt{y}}{||\mathtt{x}|| \times ||\mathtt{y}||})$, where $\mathtt{x} \cdot \mathtt{y}$ is the inner product of $\mathtt{x}$ and $\mathtt{y}$, and $||\mathtt{x}||$ is the length (i.e., the Euclidean distance of $\mathtt{x}$ from the origin) of $\mathtt{x}$. It is shown that angular distance is more suitable than Euclidean distance for high dimensional data as angular distance better reflects the contrast between near and far points [18, 42]. See § 1.4.2 for a discussion on the limitations of utilizing the Euclidean distance on high dimensional data.

1.3.3 Cluster Analysis

Cluster analysis, or clustering, is the task of partitioning a given set of items into clusters, where within a cluster items are more *similar* to each other than to those in other clusters [2, Ch. 11-16]. More specifically, the task is to assign the same clustering *label* to the items in the same cluster, but different from those in other clusters.

There are numerous data clustering approaches targeting different purposes and applications. For instance, certain clustering algorithms rely on cost function optimization, where the cost function is formulated based on the requirements of a specific problem. One of the most famous algorithms of this type is *K-means* [43]. Given $\mathtt{K}$, prior knowledge on the number of clusters, the goal of K-means is to partition the data into $\mathtt{K}$ clusters, where each point belongs to the cluster with the closest centroid. In addition to requiring prior knowledge on the number of clusters, K-means is computationally difficult, and it is sensitive to noise and outlier points. On the other hand, *graph-based* approaches are usually more robust to noise and outlier points. In spectral clustering [44], for instance, data is modeled via a graph whose nodes are the samples and the edges denote the corresponding similarity values. The goal of

spectral clustering is to *cut* the graph into communities with maximum inter-community similarity and minimum cross-community similarity. Nevertheless, spectral clustering methods are not suitable for large volumes of data as, in general, their computational complexity is cubic in the number of data points [45]. Considering the scope of the thesis regarding the challenging aspects of data (mentioned in § 1.2.1) and also the desirable applications (such as object detection using LIDAR data), the thesis considers the following two versatile clustering algorithms.

Distance-based Clustering. The items grouped together in a distance-based cluster satisfy some distance-related criteria. For example, the Euclidean clustering algorithm [5,6] partitions a given point cloud into an a priori unknown number of clusters. It works with two adjustable parameters: `minPts` and ϵ. The algorithm produces clusters each containing at least `minPts` number of points, and within each cluster, each point lies in the ϵ-radius neighbourhood of at least another point in the same cluster. Points not belonging to any cluster are characterized as *noise*. Lisco [46] is a single-pass continuous version of the Euclidean clustering algorithm designed for sorted data; e.g., a point cloud gathered by a LIDAR sensor is angularly sorted because the point cloud is collected by the spinning head of the sensor.

Density-based Clustering. The density-based clustering algorithms partition a given point-cloud into high density regions (clusters), separated by contiguous regions of low density regions. For example, Density-Based Spatial Clustering of Applications with Noise, or DBSCAN [47], is a *density-based* clustering algorithm that partitions a given point-cloud into an a priori unknown number of clusters. Similar to the Euclidean clustering algorithm, DBSCAN works with two adjustable parameters: `minPts` and ϵ. A Cluster found by DBSCAN consists of at least one *core point* and all the points that are *density-reachable* from it. Point p is a core point if it has at least `minPts` points in its ϵ-radius neighbourhood. Point q is *directly reachable* from p if q lies in the ϵ-radius neighbourhood of p. Point q is density-reachable from p, if q is directly reachable either from p or another core point that is density-reachable from p. Non-core points that are not density-reachable from any core-points are outliers [20]. DENCLUE [48], STING [49], and OPTICS [50] are some other well-known density-based clustering algorithms.

Problem Description

Given an input dataset `D`, neighbourhood radius ϵ, threshold value `minPts`, the goal is to use approximation to efficiently partition `D` into an a priori unknown number of disjoint clusters according to the requirements of either Euclidean clustering or DBSCAN. In case of DBSCAN, the clustering may be performed with distance measures other than Euclidean distance. Dataset `D` might be available in a central location, or it might as well be spread among many edge devices.

Metrics of Interest

The following explains the evaluation metrics used in this thesis to compare the proposed approximate methods with the exact ones.

Accuracy. Given the clustering outcomes of two clustering methods (one exact and one approximate) on the same data, the thesis utilizes *rand index* [51] to measure the similarity of the two clustering outcomes. Concretely, rand index measures the ratio of the number of pairs of elements that are either clustered together or separately in both clusterings, to the total number of pairs of elements.

Computational Complexity and Latency. Given a clustering algorithm, the thesis measures its computational complexity with respect to number of data points, dimensionality, number of workers (e.g., threads, fog/edge devices), and other related parameters. The thesis also empirically measures the computational latency in terms of elapsed real time.

1.3.4 Locality-Sensitive Hashing (LSH)

Locality-sensitive hashing is a relatively recent and established approach for approximate similarity search, based on the idea that if two data points are considered close, their hashed values are equal with high probability. Similarly, if two points are considered far apart, their hashed values are non-equal with high probability [19, 29, 52]. LSH can be used for low and high dimensional data, and appropriate LSH functions have been suggested for the Euclidean distance, angular distance, hamming distance, and etc.

Let $\mathtt{S}$ be the domain of the data points, and let `Distance` be the distance measure required by the applications. A *family of LSH functions* $\mathcal{H}$= $\{\mathtt{h} : \mathtt{S} \rightarrow \mathtt{U}\}$ is $(\mathtt{d}_1, \mathtt{d}_2, \mathtt{p}_1, \mathtt{p}_2)$-sensitive for distance measure `Distance` if for any $\mathtt{p}$ and $\mathtt{q}$ in $\mathtt{S}$ the following conditions hold:

- if $\texttt{Distance}(\mathtt{p}, \mathtt{q}) \leq \mathtt{d}_1$, then $\Pr_{\mathcal{H}}[\mathtt{h(p)} = \mathtt{h(q)}] \geq \mathtt{p}_1$
- if $\texttt{Distance}(\mathtt{p}, \mathtt{q}) \geq \mathtt{d}_2$, then $\Pr_{\mathcal{H}}[\mathtt{h(p)} = \mathtt{h(q)}] \leq \mathtt{p}_2$,

where the probabilities are stated over the random choices in $\mathcal{H}$. Figure 1.1 illustrates the $(\mathtt{d}_1, \mathtt{d}_2, \mathtt{p}_1, \mathtt{p}_2)$-sensitivity.

The following introduces LSH families for the Euclidean and the angular distances.

LSH Family for the Euclidean Distance. For a randomly chosen $\mathtt{u}$ in domain $\mathtt{S}$, $\mathtt{h}_\mathtt{u}(\mathtt{x}) = \lfloor \frac{\mathtt{x} \cdot \mathtt{u}}{\epsilon} \rfloor$ is an LSH function, where $\lfloor . \rfloor$ is the floor function, $\cdot$ is the inner product, and ϵ is an arbitrary constant[1]. The family is applicable for any number of dimensions. In a 2-dimensional domain, it is $(\epsilon/2, 2\epsilon, 1/2, 1/3)$-sensitive [18]. Intuitively, for a given LSH function $\mathtt{h}_\mathtt{u}(.)$, the hash value of $\mathtt{x}$ is the id of the *bucket* in which the orthogonal projection of $\mathtt{x}$ on $\mathtt{u}$ is located, where the width of the buckets is ϵ.

[1] In clustering, this constant can be chosen in accordance to the clustering parameter ϵ.

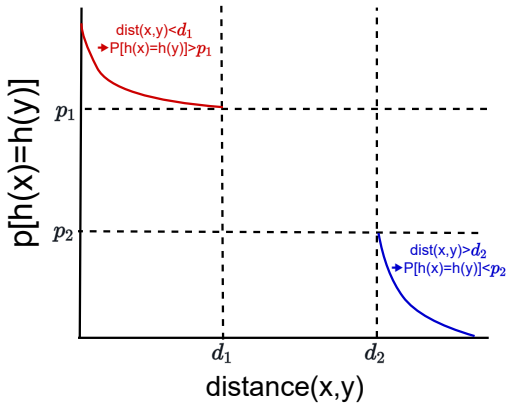


Figure 1.1: Visual illustration of $(\mathtt{d}_1, \mathtt{d}_2, \mathtt{p}_1, \mathtt{p}_2)$-sensitivity for a family of hash functions.

LSH Family for the Angular Distance. For a randomly chosen $\mathtt{u}$ in domain $\mathtt{S}$, $\mathtt{h}(\mathtt{x}) = \mathtt{sgn}(\mathtt{x} \cdot \mathtt{u})$ is an LSH function, where $\mathtt{sgn}$ is the sign function. This family is $(\theta_1, \theta_2, 1 - \frac{\theta_1}{\pi}, 1 - \frac{\theta_2}{\pi})$-sensitive, where θ_1 and θ_2 are any two angles (in radians) such that $\theta_1 < \theta_2$ [53]. Intuitively, for a given LSH function $\mathtt{h}_\mathtt{u}(.)$, the hash value of $\mathtt{x}$ indicates whether $\mathtt{x}$ is located on the *positive* or *negative* side of the hyper-plane defined by normal vector $\mathtt{u}$.

Amplifying an LSH Family

The following explains how the sensitivity of LSH functions can be *amplified* [18].

***AND*-construction.** This construction creates a new LSH family where each member is composed by independently choosing $\mathtt{M}$ arbitrary functions in $\mathcal{H}$, namely h_1, h_2, $\cdots$, $h_\mathtt{M}$. With the new hash function, $\mathtt{p}$ and $\mathtt{q}$ get hashed to the same bucket if $h_i(\mathtt{p})$ is equal to $h_i(\mathtt{q})$ for all $\mathtt{i} \in \{1, \cdots, \mathtt{M}\}$. The resulting LSH family is $(\mathtt{d}_1, \mathtt{d}_2, \mathtt{p}_1{}^\mathtt{M}, \mathtt{p}_2{}^\mathtt{M})$-sensitive.

***OR*-construction.** Similar to *AND*-construction, *OR*-construction creates a new LSH family hash function where each member is composed by independently choosing $\mathtt{L}$ arbitrary functions in $\mathcal{H}$, namely h_1, h_2, $\cdots$, $h_\mathtt{L}$. Nevertheless, in this case, $\mathtt{p}$ and $\mathtt{q}$ get hashed to the same bucket if $h_i(\mathtt{p})$ is equal to $h_i(\mathtt{q})$ for at least one $\mathtt{i}$. The resulting LSH family is $(\mathtt{d}_1, \mathtt{d}_2, 1 - (1 - \mathtt{p}_1)^\mathtt{L}, 1 - (1 - \mathtt{p}_2)^\mathtt{L})$-sensitive.

***Cascading* the two Constructions.** The aforementioned constructions can be arbitrarily cascaded in order to create new LSH families with different sensitivities, suitable for different applications. Figure 1.2 shows the probability of two given points $\mathtt{p}$ and $\mathtt{q}$ being hashed to the same bucket as a function of their angular distance using a variety of cascading, where the AND construction is applied first, and then cascaded with an OR construction.

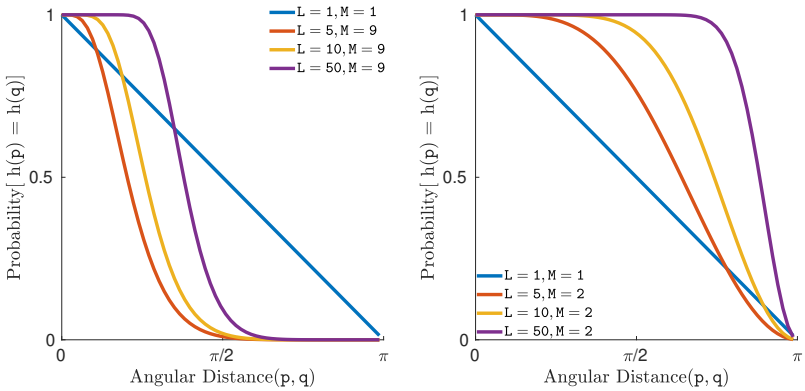


Figure 1.2: The probability of two given points p and q being hashed to the same bucket as a function of their angular distance using a variety of cascading, where the AND construction is applied first, and then cascaded with an OR construction.

1.4 Research Challenges

This thesis studies the problem of efficient big data clustering and its applications as a representative problem in IoT-based systems. To that end, the thesis explores the associated challenges of utilizing the scale down, scale up, and scale out guidelines [24]. Particularly, the thesis looks into the challenges of achieving high efficiency and scalability by integrating approximation (as a scale down approach) into the algorithmic design of distributed and parallel data clustering algorithms. The thesis also studies the challenging application-related aspects of data clustering. For example, (i) the problem of object detection and localization using distributed LIDAR point clouds in an environment scanned by several LIDAR sensors, (ii) the problem of *geofencing* (i.e., detecting objects within a restricted area in an environment), and (iii) the problem of route analysis using geolocation data. § 1.4.1 and § 1.4.2 outline the challenges of efficient distributed and parallel data clustering, respectively.

1.4.1 Distributed Data Clustering with Fog/Edge Devices

Combining readings from multiple sensors is commonly recognized as *sensor fusion*. In certain scenarios it is useful to have *multiple* sensors scan the environment simultaneously. For example, a LIDAR sensor installed in a particular place in the environment might have an *occluded* view. In order to address the occlusion [54] and increase safety and robustness, multiple LIDAR sensors can be installed in different locations to scan the environment from different perspectives. However, leveraging a distributed network of sensors induces further challenges, explained in the following.

Challenges. Centrally gathering all the local sensor data into a single source and performing the clustering algorithm on their union, known as the *fused data*, is cumbersome. Firstly, transmitting multiple sources of data (each containing hundreds of thousands or even millions of points) imposes high latency and takes long time [55, 56], as long as a few tens of seconds. Furthermore,

concurrent transmission of large volumes of data is not scalable with increasing number of devices because at some point the communication channel throttles and may collapse [15, 16]. Despite the advancements in the field of data communication, the aforementioned issues will continue to be problematic because the data collection rates of IoT sensors keep rising as well [57]. Moreover, computationally speaking, performing the clustering algorithm on the fused data takes prohibitive amount of time on the fog/edge devices because of the large volumes of the cumulative data.

Research Questions

The aforementioned challenges arise from centrally gathering distributedly gathered data. On the other hand, considering the distributed nature of the problem, the following research questions arise.

- Q_1. How to distribute computational workload across cloud, fog, and edge devices to optimize communication and computation overheads?
- Q_2. How can approximation-based synopsis help balance the required communication and computational overheads with respect to accuracy?

1.4.2 Parallel Data Clustering on Shared Memory Multi-Core Systems

In addition to distributed data clustering, the thesis also examines the problem of efficient data clustering using shared memory multi-core systems. To that end, the thesis explores the possibilities of scaling up the computation [24] on a single node (as noted in § 1.2.2) in form of parallelization in a wide range of settings. The following categorizes the associated challenges.

Parallelization Challenges. A parallel clustering algorithm employing several workers imposes the following challenges: (i) the distribution of the workload among the workers, (ii) how each worker processes its assigned workload, and (iii) synchronization and communication among the workers. To gain efficiency and scalability, the aforementioned challenges must be addressed in a way to decrease the contention for the shared resources and the crosstalk among each pair of workers, as suggested by USL, see § 1.2.2.

Data-driven Challenges. Neighbourhood and range queries are extensively required by the clustering algorithms that the thesis studies (see § 1.3.3). To avoid expensive brute-force approaches, indexing data structures can be utilized. For example, KD-trees [58] is a space partitioning tree where each level partitions the data along one of its dimensions, and different levels alternate between different dimensions. However, the majority of commonly used indexing data structures (including KD-trees [58], Octrees [59], and voxel grids [5]) grow in size with respect to the size of dataset. Furthermore, the cost of common queries and operations can become expensive (e.g., super linear in the number of data points). Skewed data distributions can also critically challenge the performance of such data structures [21–23, 60]. For example, Figure 1.3 visualizes the structure of a balanced KD-tree (where each partition

divides the data into two equal parts) on a highly skewed two-dimensional data distribution. The distribution peaks at the bottom left corner, where the figure zooms-in twice. As the figure illustrates, the cell sizes vary vastly across the structure, but most points are located in very small-size (in width and height) cells. Therefore, most range and neighbourhood queries require to examine a substantial portion of the points, rendering the efficiency of the KD-tree structure similar to a brute-force approach. Furthermore, as data in different applications can exhibit different levels of spatio-temporal locality, special methods for different levels of spatio-temporal locality are needed for higher efficiency.

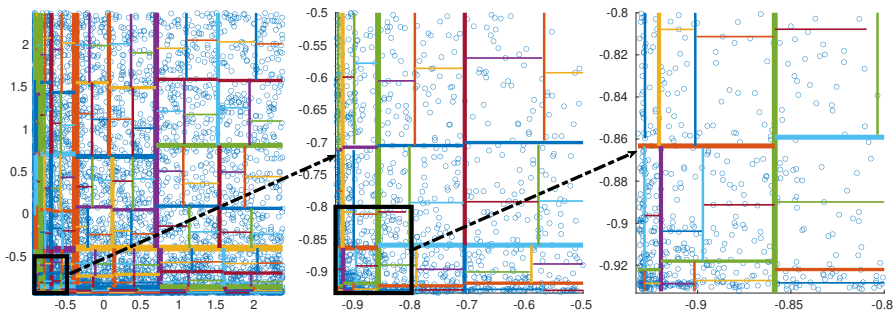


Figure 1.3: The structure of a KD-tree on a highly skewed two-dimensional data distribution. The distribution peaks at the bottom left corner, where the figure zooms-in twice. Majority of points are located in small-size cell. Therefore, most range and neighbourhood queries require to examine a substantial portion of the points, rendering the efficiency of the structure similar to a brute-force approach.

Challenges Regarding High Dimensional Data. One of the most detrimental effects of high number of dimensions is the concentration effect of L_p norms [61], which "*is the surprising characteristic of all points in a high dimensional space to be at almost the same distance to all other points in that space*" [62]. Being a special case of L_p norms when $p = 2$, the Euclidean distance is one of the most commonly used distance measures, but its usability is limited in use-cases concerning high dimensions because of the concentration effect of L_p norms. Furthermore, the computational complexity of classical neighbourhood query methods, commonly used in data processing, increases exponentially with the number of dimensions [20]. Quoting from [20] about the classical indexing structures: "*Similar to R*-trees and most index structures, grid-based approaches tend to not scale well with increasing number of dimensions, due to the curse of dimensionality [citations]: the number of possible grid cells grows exponentially with the dimensionality*". It has been shown that angular distance is a more suitable alternative for high dimensional data as it better reflects the contrast between near and far points [18, 42]. However, most challenges of efficient parallel clustering using the angular distance (and other alternative distance measures) remain unaddressed as most scientific efforts on parallel data clustering have only focused on utilization of the Euclidean distance, e.g., [7, 46, 63–66].

Research Questions

The work-sharing design of a parallel clustering algorithm distributes the workload among the workers and regulates the processing of each worker, as well as the communication and synchronization among the workers. Furthermore, the algorithmic approach of a clustering algorithm determines the level at which data-related challenges are met. Consequently, the following research questions arise regarding efficient and scalable parallel data clustering.

- Q_3. How to reduce the total workload as well as the contention and crosstalk among the workers using an approximation-based approach?
- Q_4. How to adjust the trade-off between the execution latency and the approximation accuracy?
- Q_5. How can an algorithmic approach be structured with respect to the properties of the data it has to process (e.g., high skewness, high dimensionality, and varying levels of spatio-temporal locality)?

1.5 Contributions

The thesis contributes towards efficient big data clustering and applications. To counter-balance the challenging aspects of big data (discussed in § 1.2.1) and achieve efficiency and scalability, the methods in the thesis leverage scaling down, scaling up, and scaling out (discussed in § 1.2.2) guidelines [24]. The thesis takes into account concerns regarding the design and algorithmic implementation of the methods based on the behaviour of the target platform and also the properties of the data to be processed. Regarding parallel data clustering, the thesis proposes methods that are aligned with the key data characteristics (such as skewness, spatio-temporal locality, and dimensionality) to lower the contention and crosstalk among the workers, as suggested by USL. Regarding the distributed devices in fog/edge computing, the thesis proposes methods that minimize the communication volume among the fog/edge devices, and perform the computational tasks as close as possible to the generating sources of data.

This section outlines the contributions of the thesis regarding the research questions in § 1.4. § 1.5.1 outlines the contributions of the thesis towards distributed data clustering using the scaling down and scaling out techniques. § 1.5.2 describes the contributions regarding parallel data clustering on shared-memory multi-core systems using scaling down and scaling up techniques. § 1.5.3 introduces the contributions regarding high dimensional data clustering on shared-memory multi-core systems using scaling down and scaling up techniques. § 1.6 elaborates on how the thesis advances the state-of-the-art.

1.5.1 Approximate Distributed Clustering

Targeting the problem of distributed data clustering with fog/edge devices, the thesis, in Chapter 2, proposes MAD-C, abbreviating Multi-stage Approximate Distributed Cluster-combining for Obstacle Detection and Localization.

MAD-C's Approach to Q_1. In MAD-C, the devices, distributedly and in parallel, perform the substantial portion of the required processing locally. Each device summarizes the locally detected clusters. The devices transmit their local summaries to collaboratively create a global summary of the clusters.

MAD-C's Approach to Q_2. Employing an approximation-based synopsis, each device locally approximates the results of its local processing. Afterwards, the fog/edge devices communicate and combine the approximation-based synopses in order to make a global synopsis corresponding to the clustering of the merged data. Each fog/edge device can generate its local approximate-based synopsis incrementally with only one pass over the data. As the local synopses are much smaller in size (compared to the original local data) and can be combined efficiently, MAD-C achieves significant communication and computational savings.

In addition to analytical results, empirical evaluation of MAD-C, both in simulation and a setup of fog/edge devices, show MAD-C leads to communication savings proportional to the volume of data, and it multiplicatively decreases the computation time while preserving a high accuracy. The thesis elaborates on MAD-C's capabilities for real-time distributed object detection and localization as well as geofencing.

1.5.2 Parallel Approximate Clustering

Targeting the problem of parallel data clustering on shared-memory multicore systems, the thesis, in Chapter 3, proposes PARMA-CC, a Family of Parallel Multistage Approximate Cluster Combining algorithms. Employing the same approximation-based synopsis developed for MAD-C to scale down the large volume of data, PARMA-CC algorithms facilitate leveraging an arbitrary number of workers to achieve a high degree of scalability with tunable approximation granularity.

PARMA-CC's Approach to Q_3. The approximation-based data synopsis that PARMA-CC algorithms utilize scales down the computational workload. Particularly, the thesis shows the reduction in the workload can be quadratic with respect to the approximation granularity. Furthermore, the synergy between the data structures and the workload distribution schemes in PARMA-CC reduces the contention on the shared resources and the crosstalk between pairs of workers, as suggested by USL. More specifically, PARMA-CC leverages data-parallelism and a specially designed shared data structure that facilitates work partitioning and work stealing. Furthermore, PARMA-CC's data structure supports in-place operations to eliminate the need for data copying or migration.

PARMA-CC's Approach to Q_4. PARMA-CC algorithms facilitate tuning the trade-off between the approximation quality and the required amount of workload. The latter is achieved by systematically determining the approximation granularity.

PARMA-CC's Approach to Q_5. The approximation and algorithmic approach of PARMA-CC algorithms addresses the challenges of highly skewed

data distributions by splitting the data into portions of lower density. Furthermore, PARMA-CC algorithms take specialized approaches regarding workload distribution and synchronization, targeting different levels of spatio-temporal locality. Concretely, to speed-up the computation on data exhibiting a high level of spatio-temporal locality, certain PARMA-CC algorithms utilize a data access control that takes advantage of a work-saving mechanism which proves useful on data with high levels of spatio-temporal locality. On the other hand, other PARMA-CC algorithms take on a wait-free data access pattern which proves to be more useful on data with low levels of spatio-temporal locality.

Analytical and empirical evaluations show that PARMA-CC algorithms achieve significantly higher scalability than the state-of-the-art methods while preserving a high accuracy, even on skewed data distributions.

1.5.3 Parallel Approximate Clustering for High Dimensional Data

Targeting the problem of parallel clustering on multi-core systems, the thesis, in Chapter 4, proposes IP.LSH.DBSCAN, abbreviating Integrated Parallel Density-Based Clustering through Locality-Sensitive Hashing. IP.LSH.DBSCAN fuses the data clustering process into creating a data synopsis based on locality sensitive hashing. Using the aforementioned data synopsis and through data parallelization and fine-grained synchronization, IP.LSH.DBSCAN facilitates efficient concurrent data clustering on massive data-sets, scaling with number of points and dimensions.

IP.LSH.DBSCAN's Approach to Q_3. IP.LSH.DBSCAN creates an LSH structure in which nearby points get hashed to the same *bucket* with high probability. *Fusing* the clustering process into creating the LSH structure, IP.LSH.DBSCAN utilizes the same data structure to identify and merge density-reachable points. Furthermore, the LSH structure in IP.LSH.DBSCAN facilitates in-place operations via one level of indirection (i.e., pointers). Moreover, data access and modifications are largely performed in a data parallel manner, and concurrent data modifications are synchronized using fine-grained synchronization techniques. Due to the aforementioned provisions, the workload in IP.LSH.DBSCAN is almost linear in the input size, and the contention on the shared resources and the crosstalk between pairs of workers are marginal. The results show the amount of work-load in IP.LSH.DBSCAN is significantly smaller than a clustering method that just utilizes a similar LSH structure because IP.LSH.DBSCAN fuses the clustering process into creating the data synopsis.

IP.LSH.DBSCAN's Approach to Q_4. IP.LSH.DBSCAN facilitates straightforward adjustment of accuracy through tuning the LSH parameters, i.e., its underlying data synopsis. The analytical and empirical results show trade-offs between total workload and IP.LSH.DBSCAN's accuracy.

IP.LSH.DBSCAN's Approach to Q_5. IP.LSH.DBSCAN's underlying data synopsis is based on locality-sensitive hashing, which has been proved to be

useful for high dimensional data, various distributions, and a variety of distance measures.

Analytical and empirical evaluations show that IP.LSH.DBSCAN can effectively cluster various data types (e.g., low/high dimensional, low/high skewness) using a wide-range of distance measures (e.g., Euclidean distance and angular distance) with high tunable clustering accuracy. Particularly, the results show IP.LSH.DBSCAN can be up to several orders of magnitude faster than state-of-the-art methods on high dimensional data.

1.6 Advances Relative to the State of the Art

The subsequent subsections reflect on how the thesis contributes towards distributed clustering and parallel clustering, respectively.

1.6.1 Distributed Clustering

The common practice regarding the problems that concern multiple sources of data is to centrally gather and process the union of different pieces of the data. Centrally gathering and processing the accumulated data has been suggested for point clouds in [67], for instance. Nevertheless, as pointed out in § 1.4.1, when the data is gathered distributedly, centrally gathering and clustering of the data is inefficient. In such cases, distributed clustering algorithms can be leveraged. For example, DBDC [68], density-based distributed clustering, is a client-server method that approximates the clustering outcome of DBSCAN. Client nodes in DBDC locally perform operations on their data and transmit some representatives, on which the server performs some extra processing and forwards the results back to the clients. Unlike MAD-C's constant-size approximation-based synopsis, there are no guarantees on the number of representatives in DBDC. Furthermore, MAD-C nodes can operate with an arbitrary connection topology. Authors in [69] propose distributed versions of some center-based clustering algorithms (e.g., K-Means, K-Harmonic-Means, and Expectation-Maximization). In an iterative approach, each local node computes sufficient statistics for its local data and then receives and aggregates sufficient statistics from other nodes to attain a global sufficient statistics. Nevertheless, MAD-C is more efficient at communication as the transmission of data between MAD-C nodes takes place only once.

1.6.2 Parallel Clustering

The following introduces three categories of related work to PARMA-CC and IP.LSH.DBSCAN.

Grid-based Methods. Most parallel data clustering methods employ a grid-based space partitioning approach, for instance [7, 20, 63, 64]. Such approaches have critical shortcomings in a number of ways. First, the performance of grid-based approaches degrades severely with highly skewed data distributions [21–23]. On the other hand, both PARMA-CC and IP.LSH.DBSCAN perform efficiently even with highly skewed data distributions. Second, the methods in this category do not offer any systematic approach on balancing

Table 1.1: Properties of PARMA-CC and IP.LSH.DBSCAN and related clustering algorithms

	Parallel	Effective against high dimensionality	Effective against highly skewed data	Balancing workload with accuracy	Supporting an assortment of `Distance` measures	Spatio-temporal optimization
Grid-based [7, 49, 63, 64]	✓	✗	✗	✗	✗	✗
LSH-based [70, 71]	✗	✓	✗	✓	✓	✗
ρ-approximate [7, 65]	✓	✗	✗	✗	✗	✗
`PARMACC`	✓	✗	✓	✓	✗	✓
`IP.LSH.DBSCAN`	✓	✓	✓	✓	✓	✗

the workload with respect to the required accuracy, but both PARMA-CC and IP.LSH.DBSCAN facilitate tuning the workload with respect to accuracy. Third, the algorithms in this category do not offer the possibility to take advantage of the spatio-temporal locality (if such a property exists in data), but some PARMA-CC algorithms are designed to leverage the spatio-temporal locality to achieve higher efficiency. Fourth, the methods in this category are designed to utilize only the Euclidean distance. However, other distance measures (such as the angular distance) might be needed for high dimensional applications due to the concentration effect of Euclidean distance (i.e., loss of contrast between nearest and furthest points) in high dimensions. IP.LSH.DBSCAN can utilize an assortment of `Distance` measures (such as the angular distance which is a better choice for high dimensional spaces). Fifth, as the number of possible grid cells grows exponentially with increasing number of dimensions, the aforementioned methods fail to scale with increasing number of dimensions [20]. On the other hand, IP.LSH.DBSCAN can easily scale with increasing number of dimensions.

LSH-based Methods. LSH indexing can be utilized to address the shortcomings of grid-based approaches. For instance, [70, 71] are approximate sequential density-based clustering methods that use LSH for fast neighbourhood queries. Although the aforementioned methods can enjoy the versatility of LSH, they are prone to the negative effects of skewed data distributions (i.e., the number of items returned by the majority of neighbourhood queries is proportional to the size of the data when the distirbution is highly skewed). On the other hand, IP.LSH.DBSCAN fuses the natural LSH bucketization of the points into the process of density-based clustering in a concurrent fashion. Esfandiari et al. [13] propose an almost linear approximate DBSCAN that identifies core-points by mapping points into hyper-cubes and counting the points in each hyper-cube. It uses LSH to find and merge nearby core-points. IP.LSH.DBSCAN integrates core-point identification and merging in one structure altogether, leading to better efficiency and flexibility in leveraging the desired distance measure.

Other Approximate-based Methods. Besides LSH-based methods, there exists other approximate-based methods that sacrifice accuracy to gain performance. For example, ρ-approximate DBSCAN [20] produces a clustering outcome which is *sandwiched* between those of DBSCAN with density parame-

ters ϵ and $\epsilon(1+\rho)$. Nevertheless, it has been shown that exact DBSCAN is faster than ρ-approximate DBSCAN for appropriately chosen parameters [7,20]. STING [49] also approximates the clustering outcome of DBSCAN. Other approximation approaches employ sampling techniques, e.g., [72]. These approaches do not address the challenges by the curse of dimensionality, but for low-dimensional data clustering, these methods can be incorporated in PARMA-CC's approximation-based synopsis.

Table 1.1 summarizes the comparison of PARMA-CC and IP.LSH.DBSCAN with respect to the state-of-the-art methods.

1.7 Conclusions and Future Work

This thesis studies the problem of efficient data clustering in the era of big data using the processing environments in the edge-cloud computing continuum. The methods in the thesis target a wide range of challenging big data properties such as large volume, high dimensionality, high skewness, and distributed sources.

To address the challenges of efficient big data clustering, the thesis follows the scale down, scale up, and scale out guidelines [24]. To scale down the data, the thesis utilizes approximation-based data synopsis for a wide range of challenging data properties. Furthermore, the thesis proposes methods tailored for scaling up the computation in one node using parallelization and methods tailored for scaling out the computation to several edge nodes for distributed computing. The methods in the thesis integrate the approximation techniques into the algorithmic design in order to increase the synergy between the software and the processing environment.

Regarding distributed approximate clustering on edge, the thesis proposes MAD-C. In comparison with a standard baseline which centrally gathers and processes the accumulated data, MAD-C leads to communication savings proportional to the number of points and multiplicative decrease in the dominating component of the processing complexity with high clustering accuracy. Regarding parallel approximate clustering, the thesis proposes a family of clustering algorithms called PARMA-CC, which take specialized approaches regarding workload distribution and synchronization targeting different data properties. Analytical and empirical evaluations show that PARMA-CC algorithms achieve significantly higher scalability than the state-of-the-art methods while preserving a high accuracy. Regarding high dimensional data clustering, the thesis proposes IP.LSH.DBSCAN which shows great versatility and benefits in clustering different data types (e.g., low/high dimensional, low/high skewness) using a wide-range of distance measures (e.g., Euclidean distance and angular distance) with high tunable clustering accuracy. Analytical study shows IP.LSH.DBSCAN's computational complexity is almost linear, and empirical evaluations show IP.LSH.DBSCAN is several orders of magnitude faster than state-of-the-art algorithms on high dimensional data.

The thesis showcases the applications of MAD-C for efficient object detection and localization using distributed LIDAR sensors. The thesis also shows how MAD-C can be adopted for efficient geofencing problems (i.e., detecting objects within a restricted area in an environment). Furthermore, the thesis provides further use-cases of PARMA-CC algorithms for efficient query processing such

as approximating the distance of a given point to the set of detected clusters, or predicting the clustering label of new point based on the existing clusters. The contributions of the thesis are publicly available as open source software.

The methods introduced in this thesis can form a basis for a more general data processing framework. A future line of research regarding MAD-C can fuse data from various types of sources (e.g., positioning sensors) with LIDAR data, in order to increase accuracy and safety. Another future line of research is to extend the methods to predict the *type* (e.g., pedestrian, cyclist, etc) of the scene-objects. Moreover, it is worthwhile to investigate the possibilities of extending the proposed methods for stream processing applications. For instance, as the readings from consecutive rotations of a LIDAR sensor are mostly similar, it is expected that an approximation-based synopsis corresponding to one rotation of a LIDAR sensor can efficiently be adjusted for the next rotation of the LIDAR sensor. Another interesting future study regarding MAD-C can study parallelization of the workload on distributed machines (e.g., using PARMA-CC algorithms). A general line of future work regarding PARMA-CC and IP.LSH.DBSCAN is to adapt the methods for GPU enabled systems. The latter seems to be a reasonable choice specially for IP.LSH.DBSCAN which extensively utilizes hashing techniques. Furthermore, IP.LSH.DBSCAN and PARMA-CC are expected to facilitate pipeline processing in conjunction with stream-processing oriented data structures. The performance and accuracy of IP.LSH.DBSCAN can further be enhanced by incorporating probing techniques [73, 74] into the method.

Chapter 2

Distributed Approximate Clustering and Applications

MAD-C: Multi-stage Approximate Distributed Cluster-combining for obstacle detection and localization

Amir Keramatian, Vincenzo Gulisano, Marina Papatriantafilou and Philippas Tsigas

This Chapter is an adaptation of the article that appeared in the *Journal of Parallel and Distributed Computing (JPDC)*, Vol. 147, pp. 248-267, Elsevier, 2021.

A preliminary version of this article appeared in:
Euro-Par 2018: Parallel Processing Workshops, Turin, Italy, August 27-28, 2018, Revised Selected Papers, vol. 11339, pp. 312-324. Springer, 2018.

Summary

The upcoming digitalization in the context of Cyber-physical Systems (CPS), enabled through Internet-of-Things (IoT) infrastructures, require efficient methods for distributed processing of the data, that is generated by multiple sources. We address the problem of obstacle detection and localization through data clustering, which is a common component for data processing in the fusion of multiple point clouds, each obtained by a LIDAR sensor. Such sensors generate data at high rates and can rapidly exhaust traditional methods that centrally gather and process the global data. To that end, we propose MAD-C, an approximate method for distributed data summarization through clustering, that can orthogonally build on known methods for fine-grained point-cloud clustering, and synthesize a decentralized approach, which exploits the distributed processing capacity efficiently and prevents saturation of the communication network. In MAD-C, corresponding to the point-cloud gathered by each LIDAR sensor, local clusters are first identified, each corresponding to an object in the sensed environment from the perspective of the respective sensor. Afterwards, the information about each locally detected object is transformed into a data-summary, computable in a continuous manner, with constant overhead in time and space. The summaries are then combined, in an order-insensitive, concurrent fashion, to produce approximate volumetric representations of the objects in the fused data. We show that the combined summaries, in addition to localizing objects and approximating their volumetric representations, can be used to answer relevant queries regarding the relative position of the objects in environment and a geofence. We evaluate the performance of MAD-C extensively, both analytically and empirically. The empirical evaluation is performed on an IoT test-bed as well as in simulation. Our results show that MAD-C leads to (i) communication savings proportional to the number of points, (ii) multiplicative decrease in the dominating component of the processing complexity and, at the same time, (iii) high accuracy (with RandIndex > 0.95), in comparison to its baseline counterpart for obstacle detection and localization, as well as (iv) linear computational complexity in terms of the number of objects, for the geofence related queries.

2.1 Introduction

Context and Motivation of the Problem

In the context of digitalization and Internet-of-Things (IoT) infrastructures, fog/edge computing platforms are emerging distributed processing infrastructures that can address scalability issues of the cloud computing paradigm. In contrast to the latter, which assumes that data is globally gathered and analysis is performed into the cloud, edge computing refers to having data and analysis in the edge nodes; fog computing is broader and it refers to having data and analysis in intermediate processing nodes, i.e. an abstraction between the edge and the cloud.

Processing sensor data is an important problem that can benefit from such computing infrastructures. For coping with the rate and volume at which data is produced by certain types of sensors, efficient approximation approaches that summarize the data and process the summaries are required, for example in the spirit of the synopses methods introduced in [17]. The importance of data summarization and summary processing is more pronounced for processing data from high-rate sensors, such as LIDAR sensors.

LIDAR (LIght Detection And Ranging) is an accurate sensing method that gives a 3D scan of the environment in which it operates. LIDAR reads the distances by reflecting pulsed laser lights and measuring the reflected lights. A LIDAR sensor typically has a rotating head that emits several layers of laser beams, resulting in a high resolution 3D scan of the environment, possibly containing hundreds of thousands of points, by one full rotation of the head. LIDAR sensors can produce readings with rates of several MBps. A *point cloud* refers to the output of a LIDAR sensor [39].

The high resolution, accuracy, and 3D nature of the point clouds produced by a LIDAR sensor make such a sensor an attractive choice for autonomous detection and localization of objects (e.g., in environments such as industrial areas or motorways). The ability of detecting and localizing objects autonomously can be useful in numerous applications and use-cases, e.g. in autonomous driving vehicles [75].

It is likely that more than one LIDAR sensors are needed in order to scan and create detailed 3D models of complex and large environments [76]. This is needed, for instance, when LIDAR laser beams from a single sensor might not be able to reach some objects because of occlusion. Moreover, even for an object which is not occluded, the LIDAR reading can be limited due to the object's disposition and the LIDAR's location; i.e., LIDAR beams might not hit some portions of the object. Engaging several LIDAR sensors for gathering data from different points of view can help us overcome issues such as *occlusion* and *partial views* of objects. Moreover, it can increase robustness by adding redundancy to deal with failure cases. This opportunity can enhance resiliency and availability, e.g. for automated vehicles in production environments, for checking safety risks such as *geofence* detection in environments where robots and humans co-exist, to mention just a couple of possible uses.

In the following, we discuss the challenges that are introduced by a multi-LIDAR setup, in the problem of detection and localization of objects, utilizing fog and edge computing in data processing.

Challenges Relative to the State-of-the Art

Detection and localization of objects using data from several LIDAR sensors is a resource-demanding problem. Given the state-of-the-art, a method that can be considered as the *baseline*, is to (i) first gather the point clouds from the different sources (edge nodes), commonly connected through a shared wireless channel, to a fog node; (ii) afterwards apply an object detection algorithm, commonly a *clustering algorithm* [5, 6], on the union of sources' point clouds, labeling each point accordingly. However, this approach is not practical due to the cumulative data volumes, resulting in: (i) prohibitive processing costs (at least linear in the point clouds' size) even for state-of-the-art parallel clustering approaches [5, 47, 50, 63, 63, 77]; (ii) prohibitive communication bandwidth requirements; for example, simultaneous transmission of six or more high-resolution point clouds can take multiple seconds using a shared wireless communication medium of common rates [55, 56]; (iii) prohibitive latency, as a combined consequence of the above.

To cope with the aforementioned limitations, continuous data processing at the *edge* (i.e. performing distributed detection and localization for each LIDAR source) can be beneficial to overcome the aforementioned limitations. However, to that end, new aspects imply challenges in identifying a fast, continuous and distributed approach: (i) efficient representation of the local results, and (ii) efficient combination of the local results in order to generate a global result; furthermore, (iii) the representation of the local results should be accurate enough to facilitate obtaining high-quality object detection and localization and, at the same time, should be compact enough to minimize the communication footprint and the complexity of applications such as geofencing.

Contributions

We propose MAD-C, a multi-stage approximate distributed cluster-combining method for obstacle detection and localization, using point-clouds from multiple LIDAR sensors; MAD-C efficiently exploits the available decentralized processing capacity at the edge nodes and prevents saturation of the communication network. A key component of our algorithm is an efficient *summarization* method. First, MAD-C clusters each point cloud at each edge node attached to a LIDAR sensor, orthogonally building on known methods for fine-grained point-cloud clustering. Each local cluster corresponds to an object in the sensed environment from the perspective of the respective sensor (i.e. occlusion can have significant influences). Then, each edge node computes a local, constant size *geometrical summary* of each object and MAD-C lets the processing nodes *combine* their findings with those of other nodes, until a complete view is reached. The following points summarize the findings regarding the properties of MAD-C.

- We show that the aforementioned summaries are computable in a continuous way, with constant overhead in time and space and can be combined in an order-insensitive concurrent fashion, in time that depends only on the number of objects and sensors instead of their point-clouds' sizes; thus MAD-C can exploit the data parallelism in the problem. We further provide a detailed analysis of the asymptotic expected completion time of

MAD-C, considering both the communication and processing overhead, over a binary tree or a flat tree (star) network topologies connecting the edge devices.

- We provide an extensive experimental study of MAD-C, on an actual IoT testbed and on simulations, with real-world and synthetic data and varying number of nodes, to cover a wide spectrum of scenarios, including very demanding cases. Based on the results, we can conclude that the common view produced by MAD-C is close in accuracy to that of the aforementioned baseline. We also observe significant improvements in processing and communication time, which are the more important aspects for fog/edge architectures and for the usage of the algorithm in time-sensitive applications.

- Furthermore, as a usage of MAD-C, we show how to leverage its properties in order to efficiently answer queries regarding geofences, e.g. to detect objects inside restricted areas in an environment.

An overview of MAD-C and results from a preliminary experimental study via simulations were first introduced in [78]. Here we provide a more detailed description of the algorithm, including procedures to tune key parameters, as well as the analytical study of its completion time. Furthermore, we provide a more extensive empirical evaluation of MAD-C with larger data volumes, over networks of various sizes, enabling to understand the scaling properties of MAD-C. Moreover, MAD-C is evaluated not only in simulation, but also in an IoT test-bed, comprising of representative fog/edge type devices. In addition, in the present work we explain how MAD-C's summarizations can be used in other applications. Specifically, we present and analyze an application of MAD-C for the geofencing problem.

The chapter is organized as follows: We cover preliminaries in § 2.2. We present MAD-C and its algorithmic implementation aspects in § 2.3. We analyze MAD-C's time complexity in § 2.4. We present extensions and examples of further usages of MAD-C in § 2.5. We provide an empirical evaluation of MAD-C in § 2.6. We present the related-work and conclusions in § 2.7 and § 2.8, respectively.

2.2 Preliminaries

This section describes the system model, the problem description, and the baseline solution. It also provides some background for self-containment of the chapter.

2.2.1 System Model and Problem Description

We assume K (≥ 1) asynchronous, interconnected nodes in the system. A processing unit and a LIDAR sensor at a known location and pose in the environment are associated with each node. In our model, nodes can be perceived as *fog/edge* devices.

Let ptCloud_i denote the *point cloud* (a set of points in the 3D space) that the LIDAR sensor associated with the i-th node, with a full rotation

of its spinning head, collects. Also, let N_i be the number of the points in ptCloud_i. The i-th node's processing unit can process ptCloud_i locally as well as communicate raw and processed data to other nodes.

A *(local) view* refers to an individual ptCloud_i, and the *merged point cloud* is the union of all the K point clouds. We consider that the views are collected at the same time, i.e. that the combined point clouds constitute a consistent snapshot of the scene. For simplicity and w.l.o.g, we assume views are expressed in the same coordinate system; otherwise, pre-processing can transform them into a canonical system: depending on each LIDAR's disposition, a rotation matrix and a translation can be applied on its point-cloud.

The latter can be performed in conjunction with reading the points and filtering away the ground points [79], in constant time per point, as the pre-processing step. Let us introduce an example scenario in which MAD-C is deployed.

Example. Figure 2.1(a) shows a scene in which three LIDAR sensors are installed at N_1, N_2, and N_3. Figure 2.1(b–d) respectively visualizes the view of each of the 3 LIDAR sensors. Figure 2.1(e) shows the merged point cloud. Notice that (i) there is at least one object missing in each local view and (ii) the views are complementary regarding the objects that are not occluded; e.g. they display almost non-overlapping segments of the car. Therefore, engaging more nodes to collect point clouds can result in higher accuracy.

We assume the existence of a *spanning tree* for the nodes to communicate and aggregate data [80]. Each node knows its children and its parent. Let the *sink* node be the root of the connection topology tree, in charge of generating a global view from data from all the nodes. We mainly consider a shared communication medium with low-bandwidth capacity (e.g. relying on wireless communication). We first present our methods under the spanning tree and no-message-loss assumptions, for ease of the presentation. Later on, we generalize our approach using known results in distributed systems.

The *goal* is to generate a continuous stream-compliant solution that utilizes the distributed network of nodes in the system to generate, at the sink node, a *map* that:

- enumerates the objects;
- for each object, provides a representation (e.g. volumetric and/or expressed as a set of points);
- is based on the information from all the K local point clouds,
- ensures high quality of detection.

Besides detection and localization of objects, this map should be usable in scenarios that require to check conditions relative to the environment, e.g. crossing of objects and dynamically defined borders, aka *geofences*. We elaborate on the precise definition of the geofence problem in the designated section, for ease of reference and for facilitating the reading of the chapter.

The properties of interest in a solution to the problem are: *(i) low time and communication complexity* and *(ii) high accuracy* of the map. Regarding the former, we estimate the number of processing steps and the amount of

(a) 3 nodes **(b)** N1's view **(c)** N2's view **(d)** N3's view **(e)** Merged view

Figure 2.1: *A scene with three LIDAR nodes located at N_1, N_2 and N_3 along with the local views and the merged view.*

information that needs to be communicated among the nodes. Regarding the latter, we use *rand Index*, commonly used to compare two clusterings (e.g. [51]). Rand index measures how much two clusterings of a sample set agree, based on the ratio of the number of pairs of elements that are either clustered together or separately in both clusterings, to the total number of pairs of elements.

2.2.2 Background and Baseline

Performing cluster analysis on a ptCloud is a helpful technique in order to group the points in ptCloud based on the *scene object* that each point belongs (e.g., as discussed in [75]). To that end, there are several suitable clustering algorithms that can be applied on point clouds to detect the scene objects, e.g. [5, 46, 47, 50].

Euclidean clustering. As a means of detecting scene objects in a given ptCloud, this algorithm partitions ptCloud into an a priori unknown number of clusters such that each cluster has at least `minPts` number of points, and within each cluster, each point lies in ϵ-radius neighbourhood of at least another point in the same cluster, where `minPts` and ϵ are two predefined values. Non-clustered points are identified as *noise* [5]. Using a kd-tree [81], a data structure for efficient indexing and neighbourhood search queries, the Euclidean clustering algorithm's expected and worst-case time complexities are respectively $\mathcal{O}(N \log N)$ and $\mathcal{O}(N^2)$, see [5, Ch. 4], on a point cloud with N points.

Euclidean clustering is an established and widely applied method [6]. Moreover, regarding the problems that concern multiple sources of point clouds, the common practice in the literature is to perform processing on the merged point clouds, for example [67]. Simplicity and no data loss (i.e. taking every single reading into account) are the advantages of processing the merged point cloud. Given the aforementioned background, we aim for a solid baseline that provides a reliable ground-truth solution for the problem in § 2.2.1 and is an established point of reference for comparison with our proposed methods. Therefore, we consider the following baseline:

Baseline. The i-th node gathers ptCloud_i (via its attached LIDAR sensor) and, at the same time, *listens/waits* for the incoming data (point clouds) from its children (if any). It merges (in the sense of set union) ptCloud_i with all the point clouds from its children and transmits the result to its parent. This

procedure continues for all the nodes until the sink node holds the merged point cloud from all the K nodes. Finally, the sink node performs the Euclidean clustering algorithm on the merged point cloud.

Observation 2.1. *Using a kd-tree for neighbourhood search, the expected and worst-case computational complexities of the baseline method are respectively* $\mathcal{O}(N \log N)$ *and* $\mathcal{O}(N^2)$*, where* $N = \sum_{i=0}^{K-1} N_i$*, i.e., the size of the merged point cloud.*

2.3 The MAD-C algorithm

Here we describe MAD-C and how it meets the challenges and goals explained in the previous sections.

2.3.1 The Key Idea of MAD-C

In a nutshell, each node i in MAD-C clusters ptCloud_i locally and forwards compact summaries of its local clusters. Those summaries get combined with the ones of other nodes along a spanning tree, up to the sink node, which then can deliver the set of global objects. Compared to the baseline presented in § 2.2.2, MAD-C drastically reduces the volume of data to be transmitted, while it pipelines and distributes the analysis.

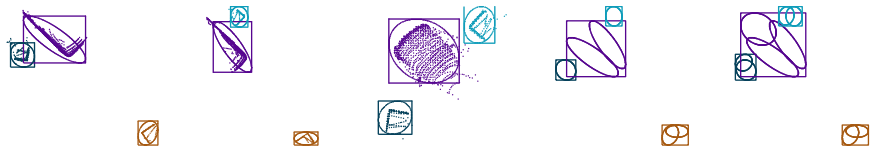
Questions arising are: how to efficiently (i) generate local maps, i.e. summaries of the local clusters in the local views; and (ii) gradually combine the maps in a deterministic fashion, despite network asynchrony. Below we elaborate on these questions and in the following subsections we explain how we treat them.

Consider two local clusters c_1 and c_2 from two distinct nodes. According to the outcome of the baseline, c_1 and c_2 constitute a bigger cluster if they overlap (or have points in the ϵ neighbourhood of each other). In that case, c_1 and c_2 should be merged. But how can we determine if c_1 and c_2 meet the conditions to be merged without calculating pairwise distances among all the points in c_1 and c_2 (i.e. the cost of a global solution)? It certainly is highly desirable if such a decision can be made with $\mathcal{O}(1)$ time complexity.

Commonly used tree-based nearest neighbour searches can not provide a solution with $\mathcal{O}(1)$ time complexity to the aforementioned problem, for instance check [18]. Furthermore, simply considering the distances between the centroids of c_1 and c_2 is not enough, because the geometrical volume that the two clusters take is also important. Therefore, we propose the MAD-C summarization technique that captures centroid, orientation, and size of each local cluster.

2.3.2 Generating Local Maps by Efficient Summarization of Local Clusters

Ideally, the summary of a local cluster c (i) uses small space (not growing with the number of points in c), (ii) can be built incrementally as new points are added in c, (iii) can be shared with peers as soon as all c's points are found, and



(a) $\mathbb{M}_1$ **(b)** $\mathbb{M}_2$ **(c)** $\mathbb{M}_3$ **(d)** $\mathbb{M}_{1,2}$ **(e)** $\mathbb{M}_{1,2,3}$

Figure 2.2: *(a),(b), and (c) are local maps. (d)* $\mathbb{M}_{1,2}$= `combine`($\mathbb{M}_1$, $\mathbb{M}_2$)*, (e)* $\mathbb{M}_{1,2,3}$= `combine`($\mathbb{M}_{1,2}$, $\mathbb{M}_3$)

(iv) expresses the geometrical *volume* that c occupies, to allow for comparisons and merging with overlapping or nearby clusters.

We identified that *bounding ellipsoids* satisfy the aforementioned requirements. Therefore, we pursue to fit a bounding ellipsoid for summarizing each local cluster. But how can we efficiently characterize the bounding ellipsoid corresponding to a local cluster? The contour surfaces of a three-dimensional Gaussian distribution are ellipsoids, where a contour surface is the set of all equi-probable points, and a three-dimensional Gaussian probability distribution is fully characterized by a mean vector $\mu \in \mathbb{R}^3$ (the centeroid of the distribution) and a covariance matrix $\Sigma \in \mathbb{R}^{3\times3}$ (the spread of the data points) [82].

The family of ellipsoids corresponding to the contour surfaces of a 3-variable Gaussian distribution are characterized by $(x-\mu)^T\Sigma^{-1}(x-\mu) = \alpha^2$, where α is a scaling factor, which we call the *confidence step*. All the ellipsoids in the family are centered at μ. With a given α, each of the three eigen-vectors of Σ defines the direction of a principal axis of the ellipsoid with the length being equal to the square root of the corresponding eigenvalue multiplied by α. [82]

Observation 2.2. *The Gaussian fit through maximum likelihood estimation [82], allows to calculate a bounding ellipsoid incrementally by calculating* $|c|$ *(*c*'s number of data points),* $S = \sum_1^{|c|} p_i$ *(cumulative vector sum of* c*'s data points) and* $\tilde{\Sigma} = \sum_1^{|c|} p_i p_i^T$ *(cumulative sum of outer products of* c*'s data points). As soon as all the points in* c *are identified,* μ *and* Σ *of the corresponding bounding ellipsoid* $\mathbb{E}$ *are calculated as* $S/|c|$ *and* $\tilde{\Sigma}/|c| - \mu\mu^T$*, respectively.*

Observation 2.3. *The representation of a bounding ellipsoid, in terms of* μ *and* Σ*, summarizing a local cluster* c *takes constant storage size, independent of* $|c|$*. Furthermore,* μ *and* Σ *can be calculated incrementally as points are being added to* c*, with constant computational overhead per point.*

Example. Figure 2.2a, 2.2b, and 2.2c respectively show the local maps corresponding to Figure 2.1b, 2.1c, and 2.1d. Note that the ellipses displayed in Figure 2.2 symbolically illustrate bounding ellipsoids summarizing local clusters. The delimiting boxes displayed in Figure 2.2 are explained in § 2.3.4.

The following definitions become useful in explaining the next steps and properties of MAD-C:

Definition 2.1. *A map* $\mathbb{M}$ *is a set of objects. An object* $\mathbb{O}$ *is a set of (bounding) ellipsoids. Two objects are* similar*, if there exists at least a pair of geometrically overlapping ellipsoids (one from each object).*

Algorithm 2.1 `generateLocalMap`(node i)

1: let $\mathbb{A}$ be the Euclidean clustering algorithm.
2: let α be the confidence step.
3: let $\mathbb{M}_i$ be an empty map.
4: **while** $\mathbb{A}$ is being applied on ptCloud$_i$
5: **if** $\exists\ p$ just clustered by $\mathbb{A}$ **then**
6: c = local cluster where p belongs
7: **if** c is new **then**
8: $|c| = 0, c.S = 0_{[3\times1]}, c.\tilde{\Sigma} = 0_{[3\times3]}$
9: $|c| = |c| + 1$; $c.S \leftarrow c.S + p$;
10: $c.\tilde{\Sigma} = c.\tilde{\Sigma} + p * p^T$
11: **for** $c \in$ local clusters detected by $\mathbb{A}$ **do**
12: $\mu = c.S/|c|$; $\Sigma = \frac{c.\tilde{\Sigma}}{|c|} - \mu\mu^T$
13: let $\mathbb{O}$ be a new object and E be a new ellipsoid
14: $E.\mu \leftarrow \mu$; $E.\Sigma \leftarrow \alpha^2\Sigma$
15: E.id $\leftarrow (i, c.\text{id})$
16: $\mathbb{O}$.ellipsoids.`add`(E)
17: **for** $d \in \{x, y, z\}$ **do**
18: $\mathbb{O}.b_d = [\min proj_d\mathbb{E}, \max proj_d\mathbb{E}]$
19: $\mathbb{M}_i$.`addObject`($\mathbb{O}$)
20: **return** $\mathbb{M}_i$

Algorithm 2.2 `unifyAtParent`(node i)

1: **for all** $j \in$ children(i) **do**
2: receive($\mathbb{M}_j$)
3: $\mathbb{M}_i \leftarrow$ `combine` ($\mathbb{M}_i$, $\mathbb{M}_j$)
4: transmit $\mathbb{M}_i$ to parent(i) (if exists)
5: `combine`($\mathbb{M}_i$, $\mathbb{M}_j$):
6: let $\mathbb{M}_{result}$ be an empty map.
7: $\mathbb{M}_{result}$.objects $\leftarrow \mathbb{M}_i$.objects $\cup\ \mathbb{M}_j$.objects
8: **for all** $\mathbb{O} \in \mathbb{M}_i$.objects, $\mathbb{O}' \in \mathbb{M}_j$.objects **do**
9: **if** overlap($\mathbb{O}.b, \mathbb{O}'.b$) **then**
10: **if** $\exists E \in \mathbb{O}$.ellipsoids $\wedge \exists E' \in \mathbb{O}'$.ellipsoids$|\mathbb{E} \cap \mathbb{E}'$ **then**
11: $\mathbb{M}_{result}$.`mergeObjects`($\mathbb{O}, \mathbb{O}'$) *with:*
12: $\mathrm{b}_d = \mathbb{O}.\mathrm{b}_d \cup \mathbb{O}'.\mathrm{b}_d | d \in \{x, y, z\}$
13: **return** $\mathbb{M}_{result}$

Algorithm 2.3 add aura δ to ellipsoid

1: `ExpandEllipoid`(Σ, δ):
2: Singular Value Decomposition: $\Sigma = V\Lambda V^T$
3: $\Lambda \leftarrow \left(\Lambda^{0.5} + \delta I\right)^2$ // I:identity matrix
4: reconstruct the matrix: $\Sigma = V\Lambda V^T$
5: **return** Σ

Stage one. The i-th MAD-C node applies the Euclidean clustering algorithm (or alternatively any relevant distance based clustering method) on ptCloud$_i$, producing *local map* $\mathbb{M}_i$. This is done distributedly, i.e. concurrently with other nodes. Procedure `generateLocalMap` in Alg. 2.1 outlines MAD-C's first stage for the i-th node. Lines 4-10 in Alg. 2.1 show how incremental ellipsoid fitting is pipe-lined into the local clustering, with constant overhead per point (see Observation 2.3). Moreover, lines 11-20 in Alg. 2.1 show how objects in $\mathbb{M}_i$ are initialized by the ellipsoids whose μ and Σ are computed using Observation 2.2. We explain how to find the value of the confidence step, α, in § 2.3.4. The highlighted lines in Alg. 2.1, correspond to algorithmic implementation details that are discussed in § 2.3.4.

MAD-C's next stage is explained in the following subsection, where we study how the maps get combined by merging the similar objects in order to generate a global map.

2.3.3 Towards a Global Map: Combining Maps

In MAD-C's second stage, each none-leaf node i receives maps from its children. It updates $\mathbb{M}_i$ by *combining* it with its children's maps and, if it is not the sink node, forwards the result to its parent.

Notice that if the combining was performed over the actual local clusters rather than their summaries, two local clusters (each one with at least $minPts$ points) would become one if at least a pair of points (one from each) is found within distance ϵ. Similarly, objects in $\mathbb{M}_i$ and each child map $\mathbb{M}_j$ are compared to detect if they are *similar* (see Definition 2.1), i.e. if they contain ellipsoids that geometrically overlap. If so, those pairs of objects are *merged*; i.e. the

union of their ellipsoids is recognized as one object in the resulting map. In § 2.3.4 we explain how (i) to integrate ϵ in an ellipsoid's representation, and (ii) to check in constant time if two ellipsoids geometrically intersect.

Notice also that applying the baseline on the merged point cloud (excluding the local noise points) results in clusters that each contain one or more local clusters from the local views. The latter holds because the points constituting a local cluster will still be clustered together (possibly with points from other local clusters) when the Euclidean clustering algorithm is applied on merged point cloud. In the same manner, the objects returned by MAD-C's sink node are sets of ellipsoids, where each ellipsoid corresponds directly to a local cluster. In other words, both methods provide a clustering of the local clusters.

Observation 2.4. *Let LC be the set of all local clusters (from different point clouds). Based on the above discussion, both baseline and MAD-C perform clustering on the elements of LC. In order to measure how much MAD-C is able to capture the clustering behaviour of the baseline, we apply a clustering similarity measurement (such as rand index) on the respective clustering outcomes of MAD-C and the baseline on the elements in LC.*

Stage two. The i-th MAD-C node executes procedure `unifyAtParent` shown in Alg. 2.2, distributedly and in parallel with other nodes in this stage. Lines 1-4 in Alg. 2.2 show that for every child node j, the i-th node receives the child map $\mathbb{M}_j$ and updates $\mathbb{M}_i$ by combining it with $\mathbb{M}_j$. If i-th node is not the sink node, when all the `combine` operations are finished, $\mathbb{M}_i$ is transmitted to the i-th node's parent. Lines 5-13 in Alg. 2.2 show how the `combine` operation is performed on $\mathbb{M}_i$ and $\mathbb{M}_j$: all pairs of similar objects (one from $\mathbb{M}_i$ and one from $\mathbb{M}_j$) are identified and merged as one object in the resulting map (the highlighted lines are implementation details covered in § 2.3.4).

Lemma 2.1. *Operation* `combine` *on maps containing ellipsoids with unique identities, satisfies the reflexive, symmetric and associative properties.*

The above follows from line 10 in Alg. 2.2, ensuring that if $\mathbb{O}_i$ and $\mathbb{O}_j$ have intersecting ellipsoids, then $\mathbb{O}_i$ and $\mathbb{O}_j$ will be merged regardless of the order of execution.

Example. Figure 2.2d shows the map resulting from `combine` ($\mathbb{M}_1$, $\mathbb{M}_2$). Figure 2.2e shows the `combine` result of the latter and $\mathbb{M}_3$.

2.3.4 Algorithmic Implementation Aspects of MAD-C

In this section, we cover three implementation-related algorithmic aspects of MAD-C.

Determining if Two Ellipsoids Geometrically Overlap. Given a pair of ellipsoids $\mathbb{E}_a, \mathbb{E}_b$, the method described in [83] determines in constant time whether they intersect. It first characterizes $\mathbb{E}_a, \mathbb{E}_b$ respectively as $X^T AX = 0$ and $X^T BX = 0$, where A and B are 4×4 matrices that (i) are derived using their centroids and covariance matrices by extending with a default row and column; and (ii) can be used to detect if there is at least an admissible

eigenvector (one that does not have a zero in the fourth dimension) of $A^{-1}B$ that satisfies both equations; in the latter case $\mathbb{E}_a, \mathbb{E}_b$ overlap.

Aura: Integrating ϵ in Ellipsoids' Representations. If the minimum distance of pairs of points from two objects is less than ϵ, then they are grouped together by the Euclidean clustering algorithm. We target the same behaviour for the ellipsoidal models, adding an *aura* $\delta = \epsilon/2$ around them. To do so, we simply increase the lengths of the main axes by this constant. Alg. 2.3 shows how to update the covariance matrix of an ellipsoid for that purpose.

Determining the Value of α, i.e. the Confidence Step. As discussed in § 2.3.2, MAD-C summarizes local clusters by bounding ellipsoids. Observation 2.2 shows how to find the parameters μ and Σ for an ellipsoid characterized by $(x-\mu)^T\Sigma^{-1}(x-\mu) = \alpha^2$, where α (the confidence step) is a scaling factor on the size of the bounding ellipsoids. The appropriate value for α is data-dependent, hence harder to estimate in a data-agnostic way. For instance, consider the bounding ellipsoids in the example in Figure 2.2. With a too small α, the bounding ellipsoids get too small to correctly cover the local clusters. On the other hand, with a too big α, the bounding ellipsoids erroneously span over other local clusters as well. Therefore, to find a proper value for α, the analyst that deploys MAD-C can apply a two-step empirical method. In the first step, the effectiveness of bounding ellipsoids for summarizing clusters can be validated for a wide range of α values. Afterwards, the clustering accuracy of MAD-C can be estimated for the α values that result in the most effective bounding ellipsoids. Finally, an α value for sample data that correspond to the deployment environment of the application, that results to the desired accuracy, can be chosen. In the following, we discuss how the effectiveness of bounding ellipsoids and the accuracy of MAD-C can be estimated.

- *Measuring Effectiveness of Bounding Ellipsoids.* For measuring the effectiveness of bounding ellipsoids, we consider the fact that MAD-C utilizes bounding ellipsoids as models summarizing clusters. To that end, *recall* and *precision* [84], two well-known model evaluation metrics, can be leveraged. For any particular local cluster c and its corresponding bounding ellipsoid e, *precision* measures the ratio of the number of points correctly covered by e to the total number of points covered by e. On the other hand, *recall* measures the ratio of the number of correctly covered points by e to the total number of points in c.

- *Measuring Accuracy.* For estimating the accuracy of MAD-C, we consider the fact that both MAD-C and the baseline perform clustering on the set of local clusters (see Observation 2.4). Therefore, to measure the agreement between the clustering outcome of MAD-C and that of the baseline, *rand index* (see § 2.2) can be employed.

The Delimiting Box Heuristic. Checking if two ellipsoids overlap (i.e. are similar) is not always necessary; such checks can be saved if e.g. they occupy distant areas of the scene. We propose *delimiting boxes* of objects to reduce the number of ellipsoid comparisons in determining if two objects are similar

(see Definition 2.1). An object's delimiting box is an axis-aligned rectangular shape that encapsulates all the ellipsoids corresponding to that object as shown in line 12 in Alg. 2.2. Similarly, an ellipsoid's *delimiting box* is the smallest axis-aligned circumscribed rectangle encapsulating that ellipsoid. Therefore, an ellipsoid's delimiting box is characterized by one closed interval on each axis as shown in line 18 in Alg. 2.1. This heuristic does not lead to any *false negatives* because if the delimiting boxes corresponding to objects $\mathbb{O}_1$ and $\mathbb{O}_2$ do not overlap, then the two objects are not similar. On the other hand, $\mathbb{O}_1$ and $\mathbb{O}_2$ are not necessarily similar if their delimiting their boxes overlap –in that case a pairwise comparison between the ellipsoids in the two objects has to be performed to determine if they are similar or not. The exact saving due to this heuristic is largely data-dependent and hence harder to estimate asymptotically, in a data-agnostic way. In § 2.6, we will empirically compare the actual number of comparisons with and without this heuristic. Please note that applying the delimiting box heuristic does not affect the validity of the properties discussed in Lemma 2.1.

2.4 MAD-C's Completion Time Analysis

In this section, we aim to characterize the asymptotic completion time behaviour of the sink node in MAD-C.

2.4.1 Assumptions, Notations, and Definitions

We assume the existence of a global clock just for the ease of exposition. Furthermore, we assume, for the local clustering, every MAD-C node leverages the Euclidean clustering algorithm employing a kd-tree for ϵ-neighbourhood search (see § 2.2.2). We assume that all MAD-C nodes start working at $t = 0$. Let MAD-C nodes be numbered $0, 1, \cdots, K-1$, and let 0 be the index of the sink node. In a binary tree topology, let $(2 \times i + 1)$ and $(2 \times i + 2)$ respectively be the indices of the left and right child of the i-th node (if applicable). For $i = \{0, \cdots, K-1\}$, let N_i be the number of points in ptCloud_i and N^* be the maximum value of N_is. Let χ_i denote the number of nodes in the sub-tree in which the i-th node is located, including the i-th node itself. For instance, χ_0 is K. Let γ denote the number of actual objects in the environment, as detected by the baseline introduced in § 2.2.2. Finally, let $\|\mathbb{M}\|$ denote the number of ellipsoids in map $\mathbb{M}$. Table 2.1 summarizes the notations that we use.

Assumption 1. *For the sake of the analysis, we assume that the asymptotic number of local clusters in each local view is* $\Theta(\gamma)$.

The above is a sensible assumption for common cases because, in each local view, while some environmental objects might be entirely occluded, others might split up into smaller ones, thus detecting $\Theta(\gamma)$ number of local clusters on average. As might be expected, this assumption is data-dependent, and there can be rare situations where the occlusion due to certain arrangement of objects in an environment increase or decrease the asymptotic number of detected local clusters.

Table 2.1: Table of Notation

K	$\triangleq$	number MAD-C nodes
γ	$\triangleq$	number of actual objects in the environment (as determined by the baseline)
α	$\triangleq$	confidence step
ptCloud_i	$\triangleq$	the i-th node's local point cloud
$\mathbb{M}_i$	$\triangleq$	the i-th node's working map
$\lvert\mathbb{M}_i\rvert$	$\triangleq$	the number of objects represented in $\mathbb{M}_i$
$\lVert\mathbb{M}_i\rVert$	$\triangleq$	total number of ellipsoids in $\mathbb{M}_i$
N	$\triangleq$	number of points in the merged point cloud ($\Sigma_{j=0}^{K-1} N_j$)
N_i	$\triangleq$	number of points in ptCloud_i
N^*	$\triangleq$	maximum number of points in all the point clouds ($\max_{j\in\{0,\cdots,K-1\}} N_j$)
χ_i	$\triangleq$	the number of nodes in the sub-tree in which the i-th node is located
$p(i,t)$	$\triangleq$	a path in the topology tree that starts from the t-th node (a leaf node) and ends at the i-th node
T_i	$\triangleq$	the completion time of the i-th MAD-C node
$\texttt{lc}_i$	$\triangleq$	the amount of time that the i-th MAD-C node spends on local clustering and creating ellipsoidal models
$\texttt{co}_i$	$\triangleq$	the amount of time that the i-th MAD-C node spends on combining its map with those of its children's
$\texttt{tr}_i$	$\triangleq$	the amount of time that the i-th MAD-C node spends on transmitting its map to its parent
$\texttt{wt}_i$	$\triangleq$	the amount of time that the i-th MAD-C node spends on waiting to receive all the maps from its children
ω_c	$\triangleq$	the execution time of each computation step
ω_t	$\triangleq$	the transmission time of each unit of data

Definition 2.2. *Regarding the i-th MAD-C node, let* $\texttt{lc}_i$ *be the amount of time that the node spends on local clustering and creating the ellipsoidal summarizations. Moreover, let* $\texttt{wt}_i$ *be the total amount of time that the node spends on waiting to receive maps from its children (if applicable). Furthermore, let* $\texttt{co}_i$ *be total amount of time that the node spends on combining its map with those of its children's (if applicable). Finally, let* $\texttt{tr}_i$ *be the amount of time that the node spends on transmitting its map to its parent (if applicable). All in all, the completion time of the i-th MAD-C node,* T_i*, is characterized as the following:* $T_i = \texttt{lc}_i + \texttt{co}_i + \texttt{tr}_i + \texttt{wt}_i$.

As the completion time of the i-th MAD-C node has computation-related and transmission-related factors, let the execution time of each computation step and the transmission time of each unit of data be denoted by ω_c and ω_t, respectively. Note that for the latter to be a constant there is a simplifying assumption that each node has dedicated bandwidth.

2.4.2 Asymptotic Behaviour of Components of the Completion Time

In this subsection, we present the asymptotic behaviour of MAD-C's completion time components. Using the latter, we will derive asymptotic bounds on the sink node's expected completion time.

Characterizing the Asymptotic Behaviour of `lc`

Lemma 2.2. *The expected asymptotic behaviour of* $\texttt{lc}_i$ *is* $\Theta(\omega_c N_i \log N_i)$*, choosing the Euclidean clustering algorithm with the kd-tree for* ϵ*-neighbourhood search as the local clustering algorithm and assuming that the resulting kd-tree is balanced and the value of* ϵ *is small enough to return constant number of points per each nearest neighbour search query.*

Proof. The expected asymptotic time complexity of the i-th node's local clustering is $\Theta(\omega_c N_i \log N_i)$ as $\Theta(N_i)$ number of nearest neighbour search queries is performed each with $\Theta(\log N_i)$ complexity. Moreover, based on Observation 2.3, the asymptotic amount of time that the i-th node spends on fitting bounding ellipsoids is $\Theta(\omega_c N_i)$. Considering the two contributing terms, the argument in Lemma 2.2 is proven. □

Characterizing the Asymptotic Behaviour of `co`

In order to analyze the asymptotic behaviour of `co`, we need to characterize the execution time of the `combine` operation first. Note that in our analysis, we do not consider the effect of the delimiting box heuristic presented in § 2.3.4 because its effect is data-dependent. Nevertheless, in practice, as we will see in the empirical evaluation section (§ 2.6), the delimiting box heuristic reduces the number of ellipsoid comparisons performed by the `combine` operation.

Lemma 2.3. *The number of ellipsoid comparisons that are performed by the* `combine` *operation on* $\mathbb{M}_i$ *and* $\mathbb{M}_j$ *is bounded by* $\Omega(\gamma^2)$ *and* $\mathcal{O}(\|\mathbb{M}_i\|\|\mathbb{M}_j\|)$ *from below and above respectively.*

Proof. $\texttt{combine}(\mathbb{M}_i, \mathbb{M}_j)$ checks the similarity (see Definition 2.1) of $\Theta(\gamma^2)$ pairs of objects because the asymptotic number of objects in each map is $\Theta(\gamma)$ based on Assumption 1.

For any given pair of objects, the similarity check immediately returns as soon it realizes whether the objects are similar or not. Therefore, in the best case, it returns with $\mathcal{O}(1)$ ellipsoid comparisons for every pair of objects. In the worst-case, however, the similarity check compares every pair of ellipsoids in $\mathbb{O}$ and $\mathbb{O}'$, in every pair of objects in $\mathbb{M}_i$ and $\mathbb{M}_j$, giving the following upper bound on the number of ellipsoid comparisons:

$$\sum_{\mathbb{O}\in\mathbb{M}_i}\sum_{\mathbb{O}'\in\mathbb{M}_j}\mathcal{O}\left(|\mathbb{O}||\mathbb{O}'|\right) = \mathcal{O}\left(\sum_{\mathbb{O}\in\mathbb{M}_i}|\mathbb{O}|\sum_{\mathbb{O}'\in\mathbb{M}_j}|\mathbb{O}'|\right) = \mathcal{O}\left(\|\mathbb{M}_i\|\|\mathbb{M}_j\|\right)$$

□

Corollary 2.1. $\Omega(\omega_c\gamma^2) \leq$ *execution time*$(\texttt{combine}(\mathbb{M}_i,\mathbb{M}_j)) \leq \mathcal{O}(\omega_c\|\mathbb{M}_i\|\|\mathbb{M}_j\|)$.

The above corollary is derived from Lemma 2.3 and noticing that comparing two ellipsoids requires $\mathcal{O}(1)$ computation steps, as discussed in § 2.3.4.

Lemma 2.4. *Let* $\mathbb{M}$ *be the* `combine` *result of two arbitrary maps* $\mathbb{M}_i$ *and* $\mathbb{M}_j$*. The number of ellipsoids in* $\mathbb{M}$ *equals the sum of number of ellipsoids in* $\mathbb{M}_i$ *and* $\mathbb{M}_j$*, i.e.* $\|\mathbb{M}\| = \|\mathbb{M}_i\| + \|\mathbb{M}_j\|$.

Proof. For any ellipsoid e in either $\mathbb{M}_i$ or $\mathbb{M}_j$, there exists an object $\mathbb{O}$ in the resulting map $\mathbb{M}$ that e is a member of $\mathbb{O}$. Conversely, the objects in $\mathbb{M}$ are composed of ellipsoids from either $\mathbb{M}_i$ or $\mathbb{M}_j$. Therefore, regardless of how

the `combine` operation merges the similar objects in the two maps, the total number of ellipsoids in $\mathbb{M}$ is always the sum of number of ellipsoids in $\mathbb{M}_i$ and $\mathbb{M}_j$. □

The following lemma presents bounds on the asymptotic behaviour of `co` for MAD-C nodes with a flat-tree (i.e. star) or binary tree connection topology.

Lemma 2.5. *For any leaf node i, $\texttt{co}_i$ is zero. Otherwise, assuming a star or binary tree connection topology, the following bounds hold on $\texttt{co}_i$: $\Omega(\omega_c\gamma^2) \leq \texttt{co}_i \leq \mathcal{O}(\omega_c{\chi_i}^2\gamma^2)$.*

Proof. For any leaf node, $\texttt{co}_i$ is zero because the leafs do not perform any combine operations. We proceed with the proof in two cases, when the connection topology is a (i) star, and (ii) a binary tree.
Star Topology. With a star topology, only the combine time corresponding to the sink node is non-zero. Therefore, let us derive $\texttt{co}_0$. The sink node sequentially combines $\mathbb{M}_0$, its map, with those of its children's. Suppose, w.l.o.g., that the sink node performs the `combine` operations in the following order: $\mathbb{M}_1, \cdots, \mathbb{M}_{K-1}$. Initially, according to Assumption 1, $\|\mathbb{M}_i\| = \Theta(\gamma)$ holds for all nodes. However, as $\mathbb{M}_0$ gets updated by each `combine` operation, $\|\mathbb{M}_0\|$ changes after each update. More specifically, after the j-th `combine` operation, $\|\mathbb{M}_0\| = \Theta(j\gamma)$, check Lemma 2.4. Therefore, applying the lower and upper bounds in Corollary 2.1, the best-case and worst-case execution time of the j-th `combine` operation are $\Omega(\omega_c\gamma^2)$ and $\mathcal{O}(\omega_c j\gamma^2)$, respectively. Summing up the lower bounds for different values of j, we get $\Omega(\omega_c K\gamma^2)$ (note that the latter is a tighter lower bound than the one introduced in Lemma 2.5). On the other hand, summing up the upper bounds for different values of j, leads us to claimed upper bound, as follows:

$$\texttt{co}_0 \leq \mathcal{O}(1\omega_c\gamma^2)+\mathcal{O}(2\omega_c\gamma^2)+\cdots+\mathcal{O}((K-1)\omega_c\gamma^2) = \mathcal{O}(\omega_c K^2\gamma^2) = \mathcal{O}(\omega_c{\chi_0}^2\gamma^2)$$

Binary Tree Topology. Let us start with deriving the lower bound. A non-leaf MAD-C node performs two combine operations, one on its left child's map ($\mathbb{M}_{2\times i+1}$) and one on its right child's map ($\mathbb{M}_{2\times i+2}$). Applying the lower bound in Corollary 2.1, $\Omega(\omega_c\gamma^2)$ is a lower bound on the execution time of the i-th node's combine operations.

Now, let us derive the upper bound. Suppose, w.l.o.g., that the i-th node first combines $\mathbb{M}_i$ with $\mathbb{M}_{2\times i+1}$ and then with $\mathbb{M}_{2\times i+2}$. The worst-case execution time of the first combine operation is bounded by $\mathcal{O}(\omega_c \frac{\chi_i}{2} \times \gamma^2)$ because at the time of combining $\|\mathbb{M}_i\|$ is $\Theta(\gamma)$, and $\|\mathbb{M}_{2\times i+1}\|$ is $\Theta(\gamma)$. However, after the first combine operation, $\|\mathbb{M}_i\|$ is $\Theta(\frac{\chi_i}{2} \times \gamma)$ (based on Lemma 2.4). Consequently, applying the upper bound in Corollary 2.1, the worst-case execution time of the second combine operation is bounded by $\mathcal{O}(\omega_c \frac{{\chi_i}^2}{4} \times \gamma^2)$, concluding the upper bound $\mathcal{O}(\omega_c{\chi_i}^2\gamma^2)$ on $\texttt{co}_i$. □

Corollary 2.2. *The following bounds hold on the combine time of the sink node in MAD-C for a star or binary tree connection topology: $\Omega(\omega_c\gamma^2) \leq \texttt{co}_0 \leq \mathcal{O}(\omega_c K^2\gamma^2)$.*

Characterizing the Asymptotic Behaviour of tr

The following lemma presents the expected transmission time for any MAD-C node.

Lemma 2.6. $\mathtt{tr}_0$ *is zero. For* $i \neq 0$*, the asymptotic behaviour of* $\mathtt{tr}_i$ *is* $\Theta(\omega_t \gamma \chi_i)$.

Proof. The first argument ($\mathtt{tr}_0 = 0$) holds because the sink node does not perform any transmissions. We prove the second argument using induction.
Base case. According to Assumption 1, the asymptotic number of local clusters in any local view is $\Theta(\gamma)$. Accordingly, the asymptotic number of bounding ellipsoids that a leaf node transmits to its parent is $\Theta(\gamma)$. According to Observation 2.3, the representation of each bounding ellipsoid takes constant size in space. Therefore, the transmission time for a leaf node asymptotically takes $\Theta(\omega_t \gamma)$. Considering that χ_i is 1 for a leaf node, the base case is proven.
Inductive step. Suppose that the argument holds for children of the j-th node, indexed by $j_1, j_2, \cdots, j_l$. The j-th node initially contains $\Theta(\gamma)$ ellipsoids in its map. However, after it performs combine operations on the maps from its children, the asymptotic number of ellipsoids in its map is $\Theta(\gamma + \gamma\chi_{j_1} + \cdots + \gamma\chi_{j_l})$, based on Lemma 2.4. On the other hand, χ_j is equal to $\chi_{j_1} + \cdots + \chi_{j_l} + 1$. Therefore, the expected asymptotic transmission time of the j-th node is $\Theta(\omega_t \gamma \chi_j)$. By mathematical induction, the argument is proven. □

Characterizing the Asymptotic Behaviour of wt

Unlike the other three time components, $\mathtt{wt}_i$, the i-th node's waiting time, can not be analytically characterized on its own because the amount of time that a node waits is execution dependent. Our next best alternative to an analytical characterization is deriving upper and lower bounds on $\mathtt{wt}_i$.

Lemma 2.7. *If the* i*-th node is a leaf node, then* $\mathtt{wt}_i$ *is zero. In general, the following bounds hold:* $0 \leq \mathtt{wt}_i \leq \max_{p(i,t)} \left(\mathtt{lc}_t + \sum_{j \in p(i,t)} (\mathtt{co}_j + \mathtt{tr}_j) \right)$, *where* $p(i,t)$ *is a path in the topology tree that starts at the* t*-th node, which is a leaf node, and ends at the* i*-th node.

Proof. The lower bound is an optimistic estimation which holds when all the i-th node's children's maps are available before the i-th node has finished its local clustering task. On the other hand, the upper bound is a pessimistic estimation that holds when the i-th node finishes its local clustering task very early and has to wait until the t-th node finishes its local clustering task. Additionally, the i-th node has to wait until all the nodes in the path $p(i,t)$ finish their combine operations and transmit the results until the i-th node receives the map from its slowest child. Therefore, the path $p(i,t)$ that maximizes $\left(\mathtt{lc}_t + \sum_{j \in p(i,t)} (\mathtt{co}_j + \mathtt{tr}_j) \right)$ determines an upper bound on $\mathtt{wt}_i$. □

Suppose $p^*(i,t)$ is the path that starts from the leaf node indexed by t^* and reaches the i-th node and maximizes the upper bound in Lemma 2.7 for the i-th MAD-C node.

Lemma 2.8. *The following bound holds for the i-th MAD-C node:*

$$\sum_{j \in p^*(i,t)} (\mathtt{co}_j + \mathtt{tr}_j) \leq \mathcal{O}(\omega_c K^2 \gamma^2) + \mathcal{O}(\omega_t K \gamma)$$

Proof.

$$\begin{aligned}
\sum_{j \in p^*(i,t)} (\mathtt{co}_j + \mathtt{tr}_j) &= \sum_{j \in p^*(i,t)} \mathtt{co}_j + \sum_{j \in p^*(i,t)} \mathtt{tr}_j \\
&\leq \left(\mathcal{O}(\omega_c K^2 \gamma^2) + \mathcal{O}\left(\omega_c \frac{K^2}{2} \gamma^2\right) + \mathcal{O}\left(\omega_c \frac{K^2}{4} \gamma^2\right) + \cdots + \mathcal{O}\left(\omega_c \frac{K^2}{2^{\lg K}} \gamma^2\right) \right) \\
&\quad + \sum_{j \in p^*(i,t)} \mathtt{tr}_j && \text{Applying Lemma 2.5} \\
&= \mathcal{O}(\omega_c K^2 \gamma^2) + \sum_{j \in p^*(i,t)} \mathtt{tr}_j && \text{Sum of the geometric series} \\
&\leq \mathcal{O}(\omega_c K^2 \gamma^2) + \left(\mathcal{O}(\omega_t K \gamma) + \mathcal{O}\left(\omega_t \frac{K}{2} \gamma\right) + \cdots + \mathcal{O}\left(\omega_t \frac{K}{2^{\log K}} \gamma\right) \right) \\
& && \text{Applying Lemma 2.6} \\
&= \mathcal{O}(\omega_c K^2 \gamma^2) + \mathcal{O}(\omega_t K \gamma) && \text{Sum of the geometric series}
\end{aligned}$$

□

Lemma 2.9. $0 \leq E[\mathtt{wt}_i] \leq \mathcal{O}(\omega_c N^* \log N^*) + \mathcal{O}(\omega_c K^2 \gamma^2) + \mathcal{O}(\omega_t K \gamma)$.

Proof. The lower bound in Lemma 2.9 is clear, so we prove the upper bound in Lemma 2.9 as the following:

$$\begin{aligned}
E[\mathtt{wt}_i] &\leq E\left[\mathtt{lc}_{t^*}\right] + E\left[\sum_{j \in p^*(i,t)} (\mathtt{co}_j + \mathtt{tr}_j) \right] \\
& && \text{Applying Lemma 2.7 and then linearity of expectation} \\
&\leq \mathcal{O}(\omega_c N^* \log N^*) + E\left[\sum_{j \in p^*(i,t)} (\mathtt{co}_j + \mathtt{tr}_j) \right] && \text{Applying Lemma 2.2} \\
&\leq \mathcal{O}(\omega_c N^* \log N^*) + \mathcal{O}(\omega_c K^2 \gamma^2) + \mathcal{O}(\omega_t K \gamma) && \text{Applying Lemma 2.8}
\end{aligned}$$

□

2.4.3 Characterizing the Completion Time T_0

Definition 2.2 explained that the i-th node's completion time comprises four components. Employing the characterizations of the components (covered in Lemma 2.2, Lemma 2.6, Lemma 2.5, and Lemma 2.9), we aim to derive lower and upper bounds on the expected completion time of the sink node.

Theorem 2.1. *The following bounds hold on the expected completion time of MAD-C's sink node with a star or a binary tree connection topology:* $\Theta(\omega_c N_0 \log N_0) + \Omega(\omega_c \gamma^2) \leq E[T_0] \leq \Theta(\omega_c N^* \log N^*) + \mathcal{O}(\omega_c K^2 \gamma^2) + \mathcal{O}(\omega_t K \gamma)$.

Proof. Based on Lemma 2.6, $\texttt{tr}_0 = 0$. Therefore, considering Definition 2.2 and the linearity of expectation, $E[T_0]$ is equal to $E[\texttt{lc}_0] + E[\texttt{co}_0] + E[\texttt{wt}_0]$. We first derive the lower bound on $E[T_0]$.

$$\begin{aligned} E[T_0] &= E[\texttt{lc}_0] + E[\texttt{co}_0] + E[\texttt{wt}_0] \\ &\geq E[\texttt{lc}_0] + E[\texttt{co}_0] && \text{Applying Lemma 2.9} \\ &\geq \Theta(\omega_c N_0 \log N_0) + \Omega(\omega_c \gamma^2) && \text{Applying Lemma 2.2 and Lemma 2.5} \end{aligned}$$

In the last statement, the best-case asymptotic behaviour of $\texttt{co}_0$ (see Lemma 2.5) is used as an asymptotic lower bound on $E[\texttt{co}_0]$. This proves the lower bound in Theorem 2.1. We now prove the upper bound on $E[T_0]$.

$$\begin{aligned} E[T_0] &= E[\texttt{lc}_0] + E[\texttt{co}_0] + E[\texttt{wt}_0] \\ &= \Theta(\omega_c N_0 \log N_0) + E[\texttt{co}_0] + E[\texttt{wt}_0] && \text{Applying Lemma 2.2} \\ &\leq \Theta(\omega_c N_0 \log N_0) + \mathcal{O}(\omega_c K^2 \gamma^2) + E[\texttt{wt}_0] && \text{Applying Corollary 2.2} \\ &\leq \Theta(\omega_c N_0 \log N_0) + \mathcal{O}(\omega_c K^2 \gamma^2) + \mathcal{O}(\omega_c N^* \log N^*) + \mathcal{O}(\omega_c K^2 \gamma^2) + \mathcal{O}(\omega_t K \gamma) \\ & && \text{Applying Lemma 2.9} \\ &= \Theta(\omega_c N^* \log N^*) + \mathcal{O}(\omega_c K^2 \gamma^2) + \mathcal{O}(\omega_t K \gamma) \end{aligned}$$

□

In practice, we expect that the number of points in each local point cloud (possibly containing hundreds of thousands of points) to be much larger than the number of nodes (K) and the number of actual objects (γ). Therefore, with small enough ω_t (i.e. fast enough transmission links), the asymptotic terms containing N_0 and N^* are the dominating factors in the expected completion time of MAD-C's sink node. In other words, we expect that either the local clustering of the sink node or one of its descendants to be the dominant factor in MAD-C's completion time.

Observation 2.5. *The longest local clustering dominates MAD-C's completion time in practice.*

2.5 Extensions and Examples of Further Usages of MAD-C

In this section, we describe how MAD-C can form the core of a set of approximations for extended usage. We also explain how MAD-C's ellipsoidal models can be further employed for efficiently processing geofence queries regarding the fusion of multiple point clouds.

2.5.1 Extensions

Versatile Communication Methods for MAD-C Nodes. Lemma 2.1 implies that the maps and the `combine` operation, satisfy the properties of *conflict-free replicated data types* [85]. Consequently, the network topology and timing asynchrony do not affect the final map at the sink node. Moreover, the

`combine` operations can be executed using non-atomic multicasting, similar to gossiping or selective flooding, guaranteeing *eventually consistent* final outcome and inherent fault-tolerance properties [80]. Therefore, the spanning tree assumption in § 2.2.1 can be lifted and besides the sink node, also any other node can construct the global map.

MAD-C-ext for Delivering Data Point Labels. MAD-C can be extended into providing a clustering label for each point in the merged point cloud, similar to what the baseline in § 2.2.2 does. The extension is as follows: Consider a leaf MAD-C-ext node indexed as i. In addition to $\mathbb{M}_i$, the i-th node transmits ptCloud_i and the corresponding local clustering labels. Its parent node, indexed as p_i, after having performed the regular tasks of a MAD-C node, relabels the points in the union of point clouds from its own and its children based on the updated $\mathbb{M}_{p_i}$. It then, in turn, transmits $\mathbb{M}_{p_i}$ along with its merged point cloud and clustering labels to its parent. As the process continues, finally the sink node holds the final merged point and its corresponding clustering labels.

Observation 2.6. *The clustering accuracy of MAD-C-ext compared to the baseline in § 2.2.2 can be computed by applying a clustering similarity measurement (such as rand index) on the clustering outcomes of MAD-C-ext and the baseline.*

2.5.2 Geofencing with the Fusion of LIDAR Point Clouds

MAD-C's ellipsoidal models can be leveraged to approximately but efficiently answer queries regarding the merged point cloud. Queries regarding geofencing are considered useful, considering that a geofence defines a predetermined perimeter in an environment marking a hazardous area for instance. We explain how the summaries produced by MAD-C, without needing to have access to the original gathered point clouds, make it possible to answer dynamically changing geofence queries. Let us give the useful definitions before we formally define the problem.

Definition 2.3 (Plane)**.** *A plane in a 3D space is characterized by* $v_1(x - x_0) + v_2(y - y_0) + v_3(z - z_0) = 0$*, where* $v = [v_1, v_2, v_3]$ *is known as the plane's* normal vector*, and* $[x_0, y_0, z_0]$ *is an arbitrary point on the plane. Any point that satisfies the characterization equation is on the plane. The plane's* positive/negative side *contains all the points that make the characterization equation greater/less than zero.*

Definition 2.4 (Geofence)**.** *A geofence is the area enclosed by the intersection of positive sides of a finite number of planes. For a geofence* `G`*, let* $|\mathtt{G}|$ *be the number of enclosing planes.*

Note that a geofence can have an arbitrary shape and size, but with a large enough number of planes any arbitrary shape can be approximated with the desired accuracy. Also note that Definition 2.4 allows a geofence to be much more general than just a regular perimeter, e.g., a polyhedron can be a geofence according to Definition 2.4. We suppose that the intersection of the positive sides of planes in `G` is non-empty.

Note that, in the literature, most geofencing problems concern positioning systems such as global navigation satellite system (GNSS) to determine whether an object has entered/exited boundaries of a geofence, for example [86]. However, not much work exists on geofencing with LIDAR data. In the following, we describe the requirements of the problem.

The geofence-crossing Problem with LIDAR Data. Given a geofence $\mathtt{G}$ and a clustering C of N data points, find out the clusters that *violate* $\mathtt{G}$, i.e. the clusters that fall inside or cross the geofence at one or more of its boundaries.

We first outline a baseline approach to address the geofencing-crossing problem, before presenting a solution based on MAD-C's ellipsoidal models.

A Baseline Solution to the Geofence-crossing Problem with LIDAR Data. For each cluster c, one can check every point in c to see whether it falls inside or crosses the geofence $\mathtt{G}$. If at least one such point is found, then the whole cluster c is identified as falling insider or crossing $\mathtt{G}$. Notice that the worst-case number of processing steps required by the straightforward solution to the geofence-crossing problem is $\mathcal{O}(N \times |\mathtt{G}|)$. The latter holds because, in the worst-case, every point has to be checked against all the $|\mathtt{G}|$ number of planes in $\mathtt{G}$, and it takes constant number of processing steps to determine to which side of a plane any given point falls.

Geofence-crossing Problem Using MAD-C

We discuss here how MAD-C's ellipsoidal summaries can be employed to efficiently find out the objects that are located inside a geofence or cross it at one or more of its boundaries, i.e. the objects that violate the geofence.

Definition 2.5. *An object $\mathbb{O}$ violates a geofence $\mathtt{G}$, if at least one ellipsoid in $\mathbb{O}$ violates $\mathtt{G}$. An ellipsoid e violates a geofence $\mathtt{G}$, if, for every plane H in $\mathtt{G}$, e either falls on the positive side of H or crosses H.*

Based on Definition 2.5, we need to be able to determine the relative position of an ellipsoid with respect to a given plane H which is either of the three following possibilities: (i) The ellipsoid is on the negative side of H. (ii) The ellipsoid is on the positive side of H. (iii) The ellipsoid intersects H. In the following we study how to determine the relative position of an ellipsoid E with respect to a given plane H.

Definition 2.6. *Given a plane H and an ellipsoid E, let points P_- and P_+ respectively represent the lower and upper bounds of the orthogonal projection of all E's points on the normal vector of H.*

Lemma 2.10. *An ellipsoid E neither crosses nor falls inside the positive side of H if and only if both P_+ and P_- are on the negative side of H.*

Proof. Consider Figure 2.3, where the relative position of ellipse E and the thick line that symbolically represents a plane are of interest. The thin line shows the plane's normal vector (v) passing through an arbitrary point x_0 on the plane. The point denoted by P_μ shows the orthogonal projection of E's centroid on the normal vector. The following three cases determine E's relative position with respect to the plane:

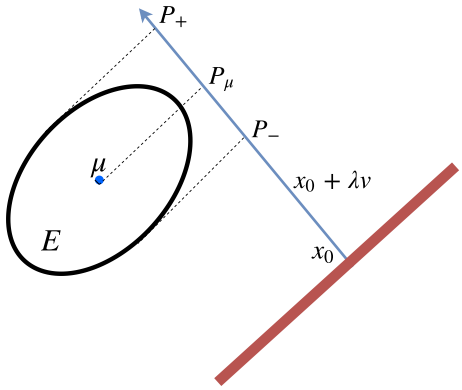


Figure 2.3: The relative position of an ellipse and the thick line that symbolically represent a plane.

- If both P_+ and P_- are on the positive side of H, then E is also located on the positive side of H.
- If P_+ is on the positive side of H, but P_- is on the negative side of H, then E intersects H.
- If both P_+ and P_- are on the negative side of H, then E is also located on the negative side of H.

□

Lemma 2.11. *All the points that fall within/on the boundaries of ellipsoid E characterized by $(x-\mu)^T\Sigma^{-1}(x-\mu) \leq 1$, are members of the following set and vice versa (the members of the following set fall within/on the boundaries of E):*

$$S_E = \{x | x = \mu + \Sigma^{(1/2)}\omega, \|\omega\| \leq 1\}$$

Proof. We note that $(x-\mu)^T\Sigma^{-1}(x-\mu)$ is equal to $\left(\Sigma^{-\frac{1}{2}}(x-\mu)\right)^T\left(\Sigma^{-\frac{1}{2}}(x-\mu)\right)$ because Σ is a symmetric positive-definite matrix. Accordingly, we can characterize the points that fall within/on the boundaries of E as the following:

$$\begin{aligned} S_E &= \left\{x \mid \left(\Sigma^{-\frac{1}{2}}(x-\mu)\right)^T\left(\Sigma^{-\frac{1}{2}}(x-\mu)\right) \leq 1\right\} \\ &= \left\{x \mid \left\|\left(\Sigma^{-\frac{1}{2}}(x-\mu)\right)\right\| \leq 1\right\} \\ &= \left\{x \mid x = \mu + \Sigma^{\frac{1}{2}}\omega, \|\omega\| \leq 1\right\} \end{aligned}$$

□

We now present a theorem that shows how the values of P_+ and P_- are calculated.

Lemma 2.12. *For a given ellipsoid E characterized by centroid vector μ and covariance matrix Σ, and a given plane H with normal vector v passing through x_0, P_+ and P_- are determined as the following: $P_\pm = \frac{v^T(\mu-x_0)}{v^Tv} \pm \left|\left|\frac{v^T\Sigma^{(1/2)}}{v^Tv}\right|\right|$.*

Proof. The orthogonal projection of any point x on the normal vector is $\frac{v^T(x-x_0)}{v^Tv}$. Applying the latter on the representation of E given in Lemma 2.11, we derive P_E, the set of points corresponding to the orthogonal projection of the points in E on the normal vector passing through x_0:

$$P_E = \frac{v^T(\mu + \Sigma^{(1/2)}\omega - x_0)}{v^Tv} = \frac{v^T(\mu - x_0)}{v^Tv} + \frac{v^T\Sigma^{(1/2)}}{v^Tv}\omega, ||\omega|| \leq 1$$

Since ω can arbitrarily be chosen from a ball with radius one, we can derive the following bounds on P_E:

$$\frac{v^T(\mu - x_0)}{v^Tv} - \Big|\Big|\frac{v^T\Sigma^{(1/2)}}{v^Tv}\Big|\Big| \leq P_E \leq \frac{v^T(\mu - x_0)}{v^Tv} + \Big|\Big|\frac{v^T\Sigma^{(1/2)}}{v^Tv}\Big|\Big|$$

With reference to the above inequalities, we conclude that $P_\pm = \frac{v^T(\mu-x_0)}{v^Tv} \pm \Big|\Big|\frac{v^T\Sigma^{(1/2)}}{v^Tv}\Big|\Big|$. □

Now that we know how to determine the relative position of an ellipsoid with respect to any given plane, we proceed with explaining our novel method for determining the objects that violate a given geofence `G`.

`ellipsoidalGeofencing`. Given a map $\mathbb{M}$ filled with MAD-C's objects, we iterate through each object $\mathbb{O}$, as shown in line 2 in Alg. 2.4, in the map to determine which ones violate the geofence `G`. To that end, if at least one ellipsoid E in $\mathbb{O}$ violates `G`, then $\mathbb{O}$ is marked as violating `G` as shown in lines 3 and 4 in Alg. 2.4.

Algorithm 2.4 Adapting MAD-C as a solution to the geofence-crossing problem with LIDAR data.

```
1:  ellipsoidalGeofencing(𝕄, G)
2:    for ∀ 𝕆 ∈ 𝕄 do
3:      if ∃ E ∈ 𝕆 | isViolating(E, G) then
4:        mark 𝕆 as violating geofence G
5:  isViolating(E, G)
6:    μ := E.μ, Σ := E.Σ
7:    for H ∈ G do
8:      Let v be the normal vector of H
9:      Let x_0 be a point on H
10:     P_+ := v^T(μ−x_0)/v^Tv + ||v^TΣ^(1/2)/v^Tv||
11:     P_− := v^T(μ−x_0)/v^Tv − ||v^TΣ^(1/2)/v^Tv||
12:     if P_+ ≤ 0 ∧ P_− ≤ 0 then
13:       return false {e neither crosses nor falls inside H}
14:   return true {E falls inside or crosses every H in G}
```

Operation `isViolating`. In order to find out if an ellipsoid E violates a geofence `G`, for evey plane H in `G`, P_+ and P_- are calculated using Lemma 2.12 as shown in lines 10 and 11 in Alg. 2.4. If the conditions in Lemma 2.10 (corresponding to line 12 in Alg. 2.4) hold for at least a plane H in `G`, then E neither crosses nor falls inside H; therefore, E does not violate `G` (see

Definition 2.5). On the other hand, corresponding to line 14 in Alg. 2.4, if E crosses or falls inside every H in $\mathtt{G}$, then E violates $\mathtt{G}$.

Lemma 2.13. *In the worst-case,* `ellipsoidalGeofencing` *asymptotically requires* $\mathcal{O}(||\mathbb{M}||\ |\mathtt{G}|)$ *processing steps.*

Proof. In the worst-case, operation `isViolating` has to be called for every ellipsoid in $\mathbb{M}$ –remember from Definition 2.1 that $||\mathbb{M}||$ denotes the total number of ellipsoids in $\mathbb{M}$. Each time, the maximum number of times that the for loop in line 7 in Alg. 2.4 gets executed is $|\mathtt{G}|$. Therefore, considering that P_+ and P_- are evaluated in constant number of processing steps, in the worst-case, `ellipsoidalGeofencing` asymptotically requires $\mathcal{O}(||\mathbb{M}||\ |\mathtt{G}|)$ number of processing steps. □

2.6 Empirical Evaluation

We empirically evaluate MAD-C and MAD-C-ext (introduced in § 2.5.1) from different perspectives. In order to perform a thorough evaluation, we conduct experiments using both a proof-of-concept implementation on an IoT test-bed, as well as a virtual-machine-based simulation. The simulation environment gives us flexibility in adjusting the parameters of the experiments to study scalability and aspects related with larger networks and more data, while the experiments in the IoT test-bed give the actual results that could be expected in a real-world MAD-C setup.

Concretely, the evaluation provides a study on (i) estimating the value of the confidence step (α), (ii) the completion time and scalability aspects of MAD-C and MAD-C-ext compared with the baseline, and (iii) the components of MAD-C's completion time. The study concerns the influence of the following parameters: the number of nodes (K), the number of scene objects (γ), and the topology of the inter-connected nodes, i.e., a star or a binary tree connection topology, as motivated in the analysis section.

Regarding (i), we use the procedure explained in § 2.3.4 to determine the value of α. Regarding (ii), we study the completion time of MAD-C in accordance with Theorem 2.1. In the same fashion, we present the completion time of MAD-C-ext, reflecting the overheads that it introduces to MAD-C. Regarding (iii), considering MAD-C's time components introduced in Definition 2.2, we study `co` (the combine time) in accordance with Lemma 2.5, and `tr` (the transmission time) in accordance with Lemma 2.6. We complement the study of the combine time by evaluating the effect of the delimiting box heuristic (presented in § 2.3.4) on the number of ellipsoid comparisons. To that end, we find the average number of ellipsoid comparisons with and without the delimiting box heuristic.

2.6.1 Evaluation Setup

We implemented MAD-C and MAD-C-ext in C++[1] and used the GNU scientific library [87] for matrix algebra. The functionalities of network communication are implemented using the Boost.Asio library [88]. For the baseline and local

[1] https://github.com/amir-keramatian/MAD-C.git

clusterings, we employed the Euclidean clustering (cf. § 2.2) algorithm in the Point Cloud Library [6], with ϵ and $minPts$ respectively set to 0.35 and 10. With these values, the baseline reasonably detects all objects in the scenes and provides a reliable ground-truth. For time measurements, we used elapsed real time.

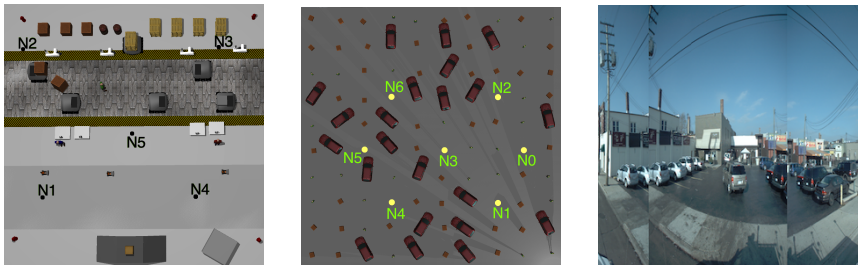
We indexed the nodes from 0 to $K-1$. The node indexed by 0 is the sink node in both topologies. Furthermore, in a binary tree topology, $(2 \times i + 1)$ and $(2 \times i + 2)$ are respectively the indices of the left and right children of the i-th node, while *height* of a node denotes the number of edges on the longest path from the node to a leaf. In the following, we give the setup details regarding the execution of MAD-C in the virtual-machine-based simulation and the IoT test-bed.

Virtual-machine-based Simulated MAD-C. On a 2.1 GHz Intel(R) Xeon(R) E5-2695 (with 3.3 GHz maximum turbo frequency) server supporting 72 threads (including hyperthreading), we emulated a range of networks sizes, ranging from 7 to 20 LIDAR sensors (i.e. MAD-C nodes), representing a reasonably high number of nodes in a realistic MAD-C deployment. Each emulated node was run inside an Oracle VM VirtualBox machine [89]. As the operating system, each virtual node ran an Ubuntu 18.04, and it was assigned 2 GB of memory and 7 hyper threads. For networking among the nodes, the virtual machines operated under the *host only adapter* option in the VirtualBox. The networking bandwidth for each simulated MAD-C node was limited to 100 Mbps via the VirtualBox settings.

MAD-C in an IoT Test-bed Consisting of Resource-constrained Nodes. These experiments, acting as both a proof-of-concept study and a validation study of the simulated systems, were run on a test-bed consisting of five ODROID-XU3 devices. Each ODROID-XU3 device is a single-board computer equipped with a Samsung Exynos 5422 Cortex-A15 2.0 GHz quad core and Cortex-A7 quad core CPUs with 2 GB of memory. Each ODROID-XU3 ran an Ubuntu 18.04 as the operating system and was connected to a switch with 100 Mbps bandwidth per port.

2.6.2 Evaluation Data

We used both real and synthetic LIDAR data sets. We generated synthetic LIDAR data sets corresponding to several scenarios, each with different characteristics in order to ensure an unbiased evaluation. The scenarios were generated by the webots simulator [90], which simulates real-world LIDAR sensors (Velodyne HDL-32E, in our case) and 3D scenes. Our first synthetic scenario resembles a factory environment with Automated Guided Vehicles, lifting arm cranes and related objects, where four LIDAR sensors are placed at the corners of the scene and one in the middle as shown in Figure 2.4a. Moreover, we generated random scenes filled with a variety of objects, as small as cubic boxes (with lengths of 80 cm) to objects as big as cars, over an area of $50 \times 50m^2$, where seven LIDAR sensors are placed in different locations as shown in Figure 2.4b. Each object is randomly rotated around its vertical axis to vary the angle with which it is exposed to the LIDAR sensors. Under



(a) *The factory scene.* **(b)** *A random scene.* **(c)** *A KITTI scene.*

Figure 2.4: Some examples of the data used in the empirical evaluation.

the aforementioned settings, we generated 300 synthetic scenarios with 10 scene objects, 300 synthetic scenarios with 50 scene objects, and 300 synthetic scenarios with 100 scene objects, respectively denoted by *syn-10*, *syn-50*, and *syn-100*.

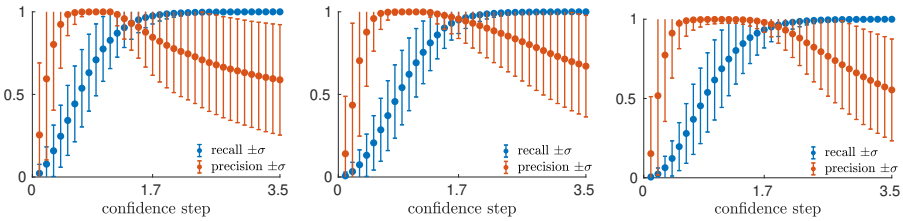
Regarding real-world LIDAR data, we utilized the Ford Multi-AV Seasonal Data Set [76], gathered by vehicles driving through the greater Detroit area, including a university campus, the DTW airport, and residential communities. Four Velodyne HDL-32E LIDAR sensors, mounted on the four corners on top of each vehicle, gathered the LIDAR data. We randomly chose a subset of the Ford Multi-AV Seasonal Data Set and split each merged point cloud into 20 overlapping partitions, reminiscent of a twenty-LIDAR sensor setup in which the sensors perceive the environment in a way that there is both redundancy and occlusion, and capture overlapping and complementary measurements. We also utilized the point clouds in the KITTI data set [91], gathered by a Velodyne HDL-64E LIDAR sensor mounted on a car driving around in urban and rural areas, see Figure 2.4c for an example. We randomly chose 43 point clouds from the KITTI data set. Since the LIDAR point clouds in the KITTI data set were gathered by a single source, we only used them to show the effectiveness of bounding ellipsoids in modelling the local clusters.

2.6.3 Evaluation Results

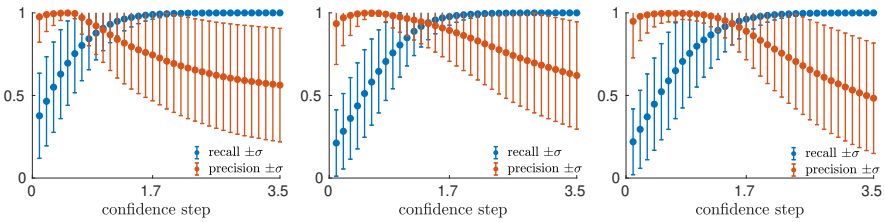
We start presenting the results by estimating the value of the confidence step.

Determining the Confidence Step

As outlined in § 2.3.4, to determine the value of α to use, the analyst that deploys MAD-C, needs to jointly tune, for representative data of the application, (i) the effectiveness of the bounding ellipsoids in summarizing the local clusters and (ii) the clustering accuracy of MAD-C, both measured for different values of α and in conjunction with the *aura* (c.f. § 2.3.4). To that end, we measured the effectiveness of the bounding ellipsoids in summarizing the clusters for a wide range of α values. Afterwards, based on the aforementioned measurements, we narrowed down the search spectrum for the suitable values of α, that result in appropriate accuracy, for sample data of each of the study cases.

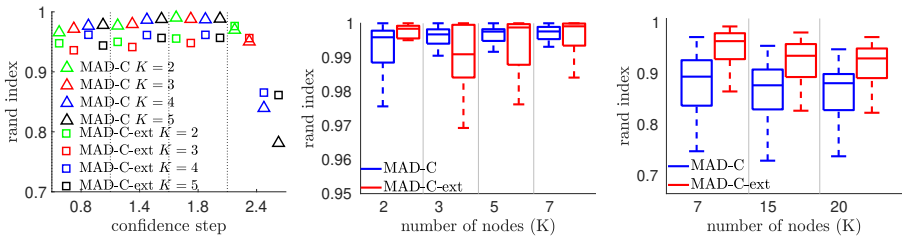


(a) Synthetic scenes, without aura **(b)** KITTI scenes, without aura **(c)** Ford Multi-AV Seasonal scenes, without aura



(d) Synthetic scenes, with aura **(e)** KITTI scenes, with aura **(f)** Ford Multi-AV Seasonal scenes, with aura

Figure 2.5: Average recall and precision with one standard deviation (confined within the zero to one interval) show the effectiveness of bounding ellipsoids in summarizing clusters as a function of the confidence step in different data sets with and without aura.



(a) Synthetic scenes **(b)** Synthetic scenes **(c)** Ford Multi-AV Data Set scenes

Figure 2.6: Accuracy of MAD-C and MAD-C-ext in rand index. 2.6a shows the clustering accuracy of MAD-C for the factory scene with different values of the confidence step and different number of nodes (i.e. K). 2.6b and 2.6c respectively show the clustering accuracy boxplots of MAD-C for the synthetic scenes and the Ford Multi-AV Seasonal Data Set for different number of nodes, with confidence step 1.5.

The Effectiveness of Bounding Ellipsoids in Summarizing Local Clusters. Plots in Figure 2.5 show the average recall and precision values with one standard deviation error bars (contained within the zero to one interval) for the different data sets, as α varies between 0.1 to 3.5. The first, the second, and the third columns in Figure 2.5 respectively correspond to the merged point clouds in the synthetic scenes, the KITTI scenes, and the merged point clouds in the Ford Multi-AV Seasonal Data Set. The first row in Figure 2.5 (2.5a, 2.5b, and 2.5c) shows the results without the aura, and the second row (2.5d, 2.5e, and 2.5f) shows the results with the aura, introduced in § 2.3.4.

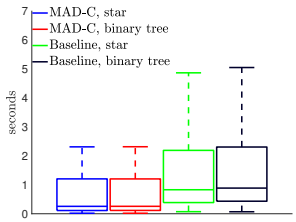
Based on the results in Figure 2.5, with a too small value for the confidence step, clusters are partly covered (i.e. low average recall) or not covered at all (i.e. low average precision). This is expected, because a small confidence step shrinks the volume of the bounding ellipsoids (remember from § 2.3.2 that the confidence step is a scaling factor on the size of the bounding ellipsoids). This problem gets mitigated as the confidence step increases; however, with a too large value for the confidence step the precision drops again because the bounding ellipsoids expand so much that they erroneously start to cover parts of other clusters as well. As the results show, adding the aura results in achieving higher recall and precision values even with relatively small values of α. The latter is expected because the aura expands the principal axis of each bounding ellipsoid by the constant $\epsilon/2$.

Furthermore, the results in Figure 2.5 show that adding the aura generally improves the effectiveness of bounding ellipsoids in summarizing the clusters because the recall and precision retain close to 1 values with more choices of α. Based on the aforementioned observations, we narrowed down the interval for the α to $[0.8, 2.4]$.

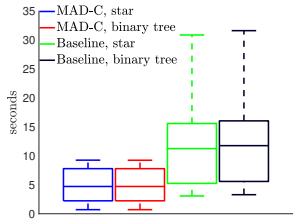
Clustering Accuracy. Figure 2.6a shows the clustering accuracy of MAD-C and MAD-C-ext on the factory scene when the number of nodes varies from two to five and the confidence step is $0.8, 1.4, 1.8$ and 2.4. We observe that rand index values are maximized when the confidence step is 1.4 and 1.8. Figure 2.6b shows the clustering accuracy of MAD-C and MAD-C-ext when confidence step is 1.5 for the synthetic scenes with two, three, four, and five nodes, where each boxplot summarizes statistics measured for syn-10, syn-50, and syn-100 data sets. Figure 2.6c shows the clustering accuracy of MAD-C and MAD-C-ext for the Ford Multi-AV Seasonal Data Set. Figure 2.6a, Figure 2.6b, and Figure 2.6c show that with an appropriate value of the confidence step, the clustering behaviour of MAD-C is similar to that of the baseline (see Observation 2.4). Figure 2.6a, Figure 2.6b, and Figure 2.6c also show that MAD-C-ext achieves high clustering accuracy (see Observation 2.6).

Completion Time of MAD-C, MAD-C-ext, and the Baseline

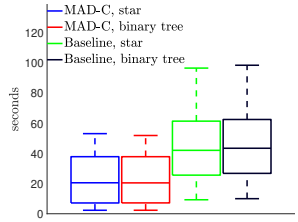
The boxplots in Figure 2.7 show the completion time (in seconds) of MAD-C and the baseline for the synthetic scenes with the star and binary tree connection topologies. The first, second, and the third columns respectively correspond to results with syn-10, syn-50, and syn-100 data sets. The first row (2.7a, 2.7b, and 2.7c) shows the results obtained on the IoT test-bed with five nodes. The second row (2.7d, 2.7e, and 2.7f) shows the results obtained



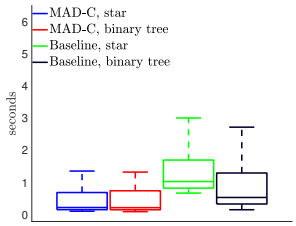
(a) syn-10, IoT test-bed with 5 nodes



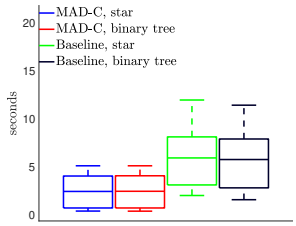
(b) syn-50, IoT test-bed with 5 nodes



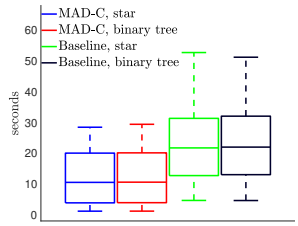
(c) syn-100, IoT test-bed with 5 nodes



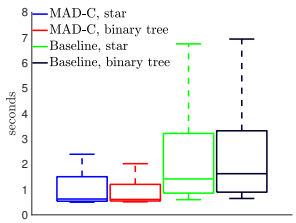
(d) syn-10, simulation with 5 nodes



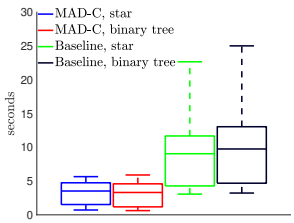
(e) syn-50, simulation with 5 nodes



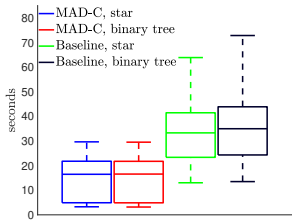
(f) syn-100, simulation with 5 nodes



(g) syn-10, simulation with 7 nodes

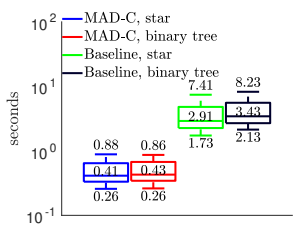


(h) syn-50, simulation with 7 nodes

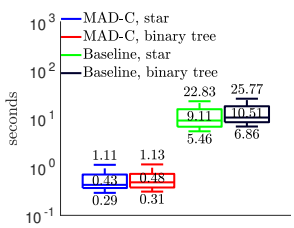


(i) syn-100, simulation with 7 nodes

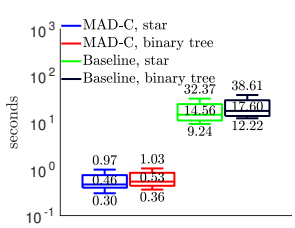
Figure 2.7: Completion time of MAD-C and Baseline in different setups for syn-10, syn-50, and syn-100 data sets. (a-c) The IoT test-bed with five nodes. (d-f) Simulation with five nodes. (g-i) Simulation with seven nodes. Note that the simulation and the IoT test-bed use different hardware platforms.



(a) Simulation with 7 nodes

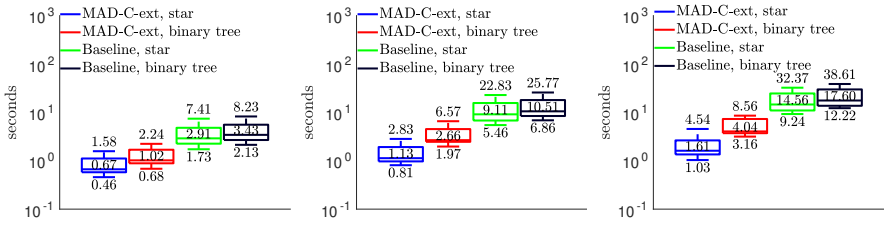


(b) Simulation with 15 nodes



(c) Simulation with 20 nodes

Figure 2.8: Completion time of MAD-C and Baseline in different setups for the Ford Multi-AV Seasonal Data Set. The lower and upper limits as well as the median values are presented for each boxplot.



(a) Simulation with 7 nodes **(b)** Simulation with 15 nodes **(c)** Simulation with 20 nodes

Figure 2.9: Completion time of MAD-C-ext and Baseline in different setups for the Ford Multi-AV Seasonal Data Set. The lower and upper limits as well as the median values are presented for each boxplot. Note that the baseline is the same as the one in Figure 2.8.

in our simulation with five nodes, and the third row (2.7g, 2.7h, and 2.7i) shows the simulation results with seven nodes. Remember from § 2.6.1 that the virtual-machine-based simulation and the IoT test-bed use different hardware platforms. The latter explains the discrepancies (in absolute times) between the simulation results with five nodes and the results of the IoT test-bed (with five nodes). Nonetheless, we observe that the ratio of completion times is consistent between the two cases. The statistical results show that MAD-C is 2-5 times faster than the baseline in the experiments with the synthetic scenes.

The boxplots in Figure 2.8 show the completion time (in seconds) of simulated MAD-C and simulated baseline for the Ford Multi-AV Seasonal Data Set with the star and binary tree connection topologies. 2.8a, 2.8b, and 2.8c respectively show the simulation results for 7, 15, and 20 nodes. In Figure 2.8, the lower and upper limits as well as the median values are presented for each boxplot to enhance readability as the scale on the Y-axis is logarithmic. The results show that MAD-C is about 5-30 times faster than the baseline, depending on the number of nodes.

Regarding *scalability* aspects, the results statistically show that, as the number of nodes increases, the increase in MAD-C's completion time is drastically lower than the increase in the baseline's completion time, as expected. For example, regarding the synthetic data set, as the number of nodes increases from five to seven, MAD-C's maximum completion time increases about 7% on syn-100 data set, considering the star connection topology. Under the same conditions, the maximum completion time of the baseline increases about 38%. Moreover, the median completion time of MAD-C remains about 0.5 seconds with 7, 15, and 20 nodes, but the median completion time of the baseline is approximately 3, 9, and 15 seconds, respectively. Besides, the results show that the completion time of MAD-C and the baseline increase on data sets containing higher number of objects, respectively confirming the analysis in Theorem 2.1 and Observation 2.1 (note that the synthetic point clouds that contain higher number of scene objects contain more number of points as well).

The growth in baseline's completion time (with respect to the number of nodes) is explained by the fact that it gathers and centrally processes all the point clouds, see Observation 2.1. On the other hand, Observation 2.5 is the critical key in understanding why MAD-C's completion time does not increase as rapidly as the baseline's does with increasing number of nodes: the length of the longest local clustering (the dominant factor in the MAD-C's completion

time, as shown in the previous section) does not grow with increasing number of nodes. As we will see in the following, transmission and combine times are negligible for MAD-C nodes.

Finally, the boxplots in Figure 2.9 show the completion time of simulated MAD-C-ext and the simulated baseline for the Ford Multi-AV seasonal Data Set with the star and binary tree connection topologies. Figure 2.9a, Figure 2.9b, and Figure 2.9c respectively show the results for 7, 15, and 20 nodes. In Figure 2.9, the lower and upper limits as well as the median values are presented for each boxplot to enhance readability as the scale on Y-axis is logarithmic. The results show that MAD-C-ext is more than three times faster than the baseline, despite the fact that MAD-C-ext transmits the raw point clouds. The savings in completion time of MAD-C-ext is due to distributing the workload among the nodes and not performing the clustering task centrally. Moreover, as presented in the following, performing combine tasks takes negligible time compared to the total completion time.

Combine Time of MAD-C Nodes

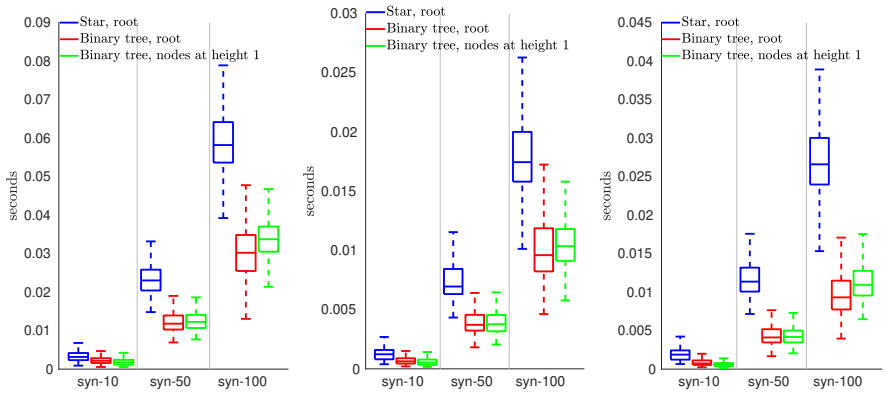
The boxplots in Figure 2.10 show the combine time of MAD-C nodes (in seconds) on syn-10, syn-50, and syn-100 data sets for the star and binary tree connection topologies. For the star topology, the boxplots show the results corresponding to the root node, which is the only node performing any `combine` operations. For the binary tree topology, the boxplots also show the results of nodes at height one (remember that leaf nodes do not perform any `combine` operations). In Figure 2.10, 2.10a, 2.10b, and 2.10c show MAD-C's combine time for the IoT test-bed (with five nodes), the virtual-machine-based simulation with five nodes, and the simulation with seven nodes, respectively.

The boxplots in Figure 2.11 show the combine time of MAD-C nodes (in seconds) on the Ford Multi-AV Seasonal Data Set for the star and binary tree connection topologies. Figure 2.11a, Figure 2.11b, and Figure 2.11c respectively show the results for 7, 15, and 20 virtual-machine-based simulated nodes, differentiating the nodes based on their height in the respective connection hierarchy.

The results in Figure 2.10 and Figure 2.11 show that, as expected, the combine time increases with increasing number of objects (γ) and increasing number of nodes (K). Moreover, for a constant K, we observe that the sink node in the star topology has higher combine time than the sink and intermediate nodes in the binary tree topology. This is expected because in the star topology, only the sink node performs `combine` operations; however, the workload gets distributed among the sink and intermediate nodes in the binary tree connection topology.

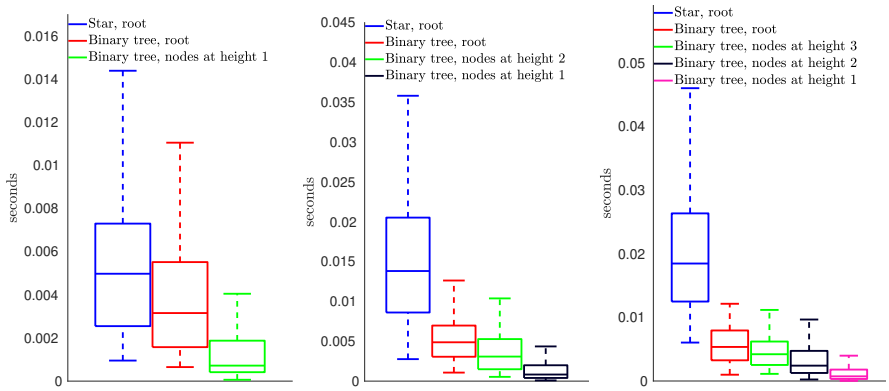
Comparing MAD-C's completion times in Figure 2.7 and Figure 2.8 with MAD-C's combine times in Figure 2.10 and Figure 2.11, we observe that the amount of time that a MAD-C node spends on combining maps is negligible compared to MAD-C's total completion time.

The following discussion examines the effect of the delimiting box heuristic (see § 2.3.4) on the average number of ellipsoid comparisons made by the `combine` operations.



(a) The IoT test-bed (5 nodes) **(b)** Simulation with 5 nodes **(c)** Simulation with 7 nodes

Figure 2.10: MAD-C's combine time in the IoT test-bed and the virtual-machine-based simulation for syn-10, syn-50, and syn-100 data sets. Note that the simulation and the IoT test-bed use different hardware platforms.



(a) Simulation with 7 nodes **(b)** Simulation with 15 nodes **(c)** Simulation with 20 nodes

Figure 2.11: MAD-C's combine time for the Ford Multi-AV Seasonal Data Set.

Table 2.2: *Average number of ellipsoid comparisons made by the* `combine` *operations with/without the delimiting-box heuristic.*

	$K = 2$		$K = 3$		$K = 5$		$K = 7$	
syn-10	16	168	46	599	92	1827	164	2951
syn-50	72	3610	218	12738	487	38858	804	56477
syn-100	105	12618	390	42481	1094	140380	2049	203423

The Effect of the Delimiting Box Heuristic. Table 2.2 shows the average number of ellipsoid comparisons made by the `combine` operations on syn-10, syn-50, and syn-100 data sets, with various number of nodes. The highlighted entries show the results when the delimiting box heuristic was employed, and the non-highlighted entries show the results without the heuristic. These results show that, with the delimiting box heuristic, the asymptotic number of ellipsoid comparisons follows a smaller growth function than the worst-case asymptotic bound presented in Lemma 2.3. Therefore, in practice, the execution time of the `combine` operation is lower than the bounds presented in Corollary 2.1.

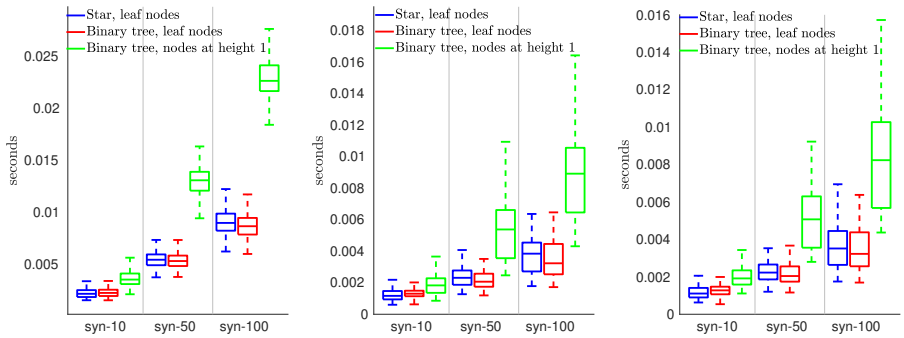
Transmission Time of MAD-C and Baseline Nodes

The boxplots in Figure 2.12 show the transmission time (in seconds) of MAD-C and baseline nodes on syn-10, syn-50, and syn-100 data sets for the star and binary tree connection topologies. For the star topology, the boxplots show the aggregate results of all the leaf nodes. For the binary tree topology, the boxplots also show the aggregate results of the intermediate nodes. The first row in Figure 2.12 (2.12a, 2.12b, and 2.12c) shows MAD-C results respectively on the IoT test-bed, the virtual-machine-based simulation with five nodes, and the virtual-machine-based simulation with seven nodes. The second row in Figure 2.12 (2.12d, 2.12e, and 2.12f) shows the results of the baseline in the same order. As explained in § 2.6.1, the nodes in the IoT test-bed are connected to a switch, but the virtual nodes in the simulation use the host machine's loopback interface with software limited bandwidth. The latter explains the discrepancies between the simulation results with five nodes and the results of the IoT test-bed (with five nodes).

The boxplots in Figure 2.13 show the transmission time (in seconds) of MAD-C and baseline nodes on the Ford Multi-AV Seasonal Data Set for the star and binary tree connection topologies, differentiating the nodes based on their height in the respective connection hierarchy. Figure 2.13a, Figure 2.13b, and Figure 2.13c respectively correspond to 7, 15, and 20 simulated MAD-C nodes. Similarly, Figure 2.13d, Figure 2.13e, and Figure 2.13f correspond to 7, 15, and 20 virtual-machine-based simulated baseline nodes, respectively.

The statistical results show MAD-C vastly cuts down on the required transmission time. For instance, comparing the results of MAD-C and the baseline on the IoT test-bed, MAD-C nodes are about 40 time faster than their baseline counterparts. Moreover, MAD-C's savings in transmission time becomes more significant with increasing number of nodes. For example, shown in Figure 2.13f, MAD-C nodes are about two orders of magnitude faster than the baseline nodes in the virtual-machine-based simulations with 20 nodes.

Based on the results in Figure 2.12, for a MAD-C node, the transmission time increases almost linearly with the number of objects (γ). Moreover, the



(a) MAD-C, IoT test-bed (five nodes)

(b) MAD-C, simulation with five nodes

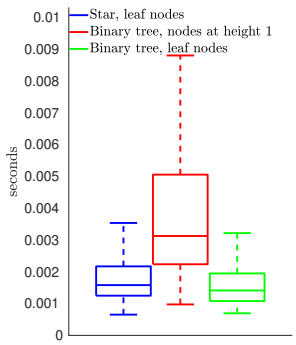
(c) MAD-C, simulation with seven nodes

(d) Baseline, IoT test-bed (five nodes)

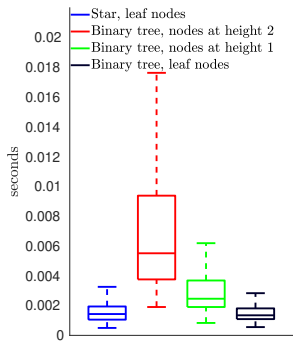
(e) Baseline, simulation with five nodes

(f) Baseline, simulation with seven nodes

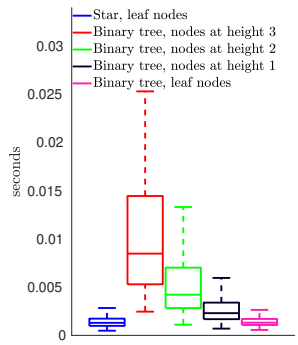
Figure 2.12: Transmission time of MAD-C and baseline nodes in different setups for syn-10, syn-50, and syn-100 data sets.



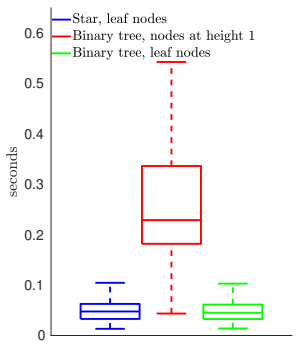
(a) MAD-C, simulation with 7 nodes



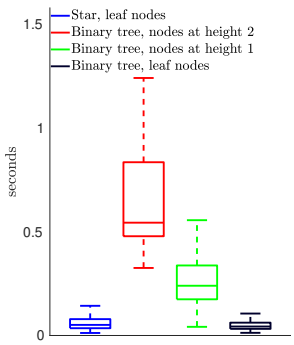
(b) MAD-C, simulation with 15 nodes



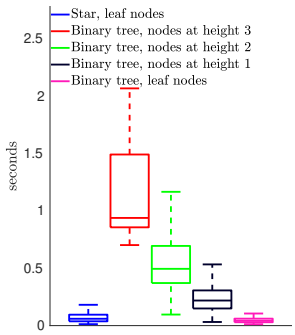
(c) MAD-C, simulation with 20 nodes



(d) Baseline, simulation with 7 nodes



(e) Baseline, simulation with 15 nodes



(f) Baseline, simulation with 20 nodes

Figure 2.13: Transmission time of MAD-C and baseline nodes in different setups for the Ford Multi-AV Seasonal Data Set.

results in Figure 2.12 and Figure 2.13 indicate that the transmission time of a node i increases with χ_i, which is the number of nodes in the sub-tree of that given node (note that the number of nodes, the topology of the tree, and the height of a given node i can determine χ_i). These observations are in accordance with the asymptotic behaviour of the transmission time provided in Lemma 2.6.

Based on the results, we expect that advantages of MAD-C over the baseline in saving transmission time become even more pronounced in setups where the communication medium is slow and subject to failures such as collision, i.e. the issues that take place in wireless communication.

Summary of the Results

In the evaluation of MAD-C, firstly we conducted experiments to empirically estimate the value of the confidence step. Then, we studied the completion time of MAD-C and MAD-C-ext and compared the results to that of the baseline. We observed the scalability of MAD-C and MAD-C-ext for varying number of nodes using real and synthetic data sets. We notably observed 2 to 30 times faster completion time with MAD-C compared to the baseline, with differences becoming more pronounced as the network sizes, the data sizes and the complexity of the sensed environment increase. Furthermore, we empirically observed that the local clustering is the dominant cost factor in MAD-C as the transmission and combine times are negligible, confirming our analytical studies. We conclude that MAD-C's completion time can be further improved by employing a faster local clustering algorithm.

2.7 Related Work

Clustering is a commonly used method to detect objects in a point cloud [5, 92]. Among the relevant clustering algorithms that can be applied on point clouds are [5, 47, 50]. As point clouds are typically large in volume, the study of clustering algorithms that utilize parallelization to tackle the challenges of clustering a large volume of data is important. For example, the method in [63] achieves parallelization by using graph algorithmic concepts. Other methods such as [93] and [94] utilize graphics processing units to achieve parallelization. As explained in the chapter, MAD-C works on top of the result of a clustering algorithm. Therefore, in deployment, MAD-C can orthogonally utilize such commodities if they are available.

Combining readings from multiple sensors is commonly recognized as *sensor fusion*. For instance, the authors in [95] propose a method for 3-D model reconstruction of objects using multiple calibrated camera views.

As mentioned in § 2.1, MAD-C and results from a preliminary experimental study via simulations were introduced in [78]. The present chapter adds the analytical study of the completion time behaviour of MAD-C. Furthermore, it provides a significantly more extensive empirical evaluation of MAD-C with binary tree and star tree connection topologies. Besides, MAD-C is evaluated here not only in simulation, but also in an IoT test-bed, comprising of representative fog/edge type devices. Furthermore, MAD-C is evaluated here using a more extensive data set. In addition, the present work explains

how MAD-C's summarizations can be used in other applications, having as an example to efficiently answer geofence queries.

Data summaries offer opportunities to efficiently deal with big volume of data generated in a streaming fashion, (cf. e.g. [96]). MAD-C is indeed a distributed algorithm that processes streaming data through data summaries.

Variants of Octrees [59], voxel grids [5], and bounding boxes [97, 98] are tools used for efficient processing of point clouds. Our work, based on bounding ellipsoids, offers new advantages because they have a compact representation, they can be calculated incrementally (as points are being added in the clusters) and they facilitate efficient operations.

Geometric alignment of two point clouds can be performed by ICP [99] when the relative location and pose of the sources is not known. In our work, however, we know the exact location and pose of all LIDAR nodes.

Distributed versions of some center-based clustering algorithms (for instance K-Means, K-Harmonic-Means, and Expectation-Maximization) are proposed in [69]. The basic iterative idea is to have the distributed nodes compute sufficient statistics for their local data and then have the local sufficient statistics aggregated to attain the global sufficient statistics which gets broadcasted back to the distributed nodes. Enough iterations of the explained procedure takes place until some predetermined performance criteria is met.

DBDC [68], density-based distributed clustering, is a client-server approach to distributed density based clustering. Firstly, each node locally performs DBSCAN. Afterwards, a small number of representatives are chosen to summarize each local cluster. The representatives from each site are then transmitted to a central coordinator for further processing, where the central coordinator creates a global clustering based on the representatives using DBSCAN with adjusted parameters. The global clustering model is subsequently transmitted to the local sites, where cluster relabeling takes place according to the global clustering model. Unlike MAD-C, there are no guarantees on the size of summaries in DBDC, i.e. the number of representatives for each local cluster can become large. Whereas DBDC only works with a client-server topology, MAD-C operates with any given connection topology. MAD-C is a distance-based clustering method, but DBDC is a density-based clustering method; therefore, they are complementary approaches.

2.8 Conclusions

MAD-C is an approximate method for distributed obstacle detection and localization in the fusion of several point clouds generated by different nodes, where each node has a LIDAR sensor and a processing unit. Each MAD-C node, distributedly and in parallel with other nodes, first applies a distance-based clustering algorithm on its local point cloud and summarizes each local cluster into a constant-sized representation. The summarization can be pipe-lined into the local clustering process with constant computational overhead per data point. After the summarizations, the nodes share their summaries and collaboratively combine them in an order-insensitive concurrent fashion, to generate a global summary corresponding to the fusion of all local point clouds. Compared to a baseline method that centrally gathers and clusters all the

point clouds, MAD-C drastically reduces the amount of information that needs to be shared among nodes. It also distributes the computational complexity among several nodes. One important usage of MAD-C is to efficiently detect objects inside a geofence, for instance an enclosed hazardous area specified in the environment.

As we saw analytically and empirically, MAD-C has smaller completion time than the baseline: MAD-C drastically cuts down on transmission volume, and its processing overheads are marginal. Regarding scalabiliy, we saw that MAD-C's benefits become more pronounced as the number of nodes and the complexity of the scenes increases. We also noticed and discussed that the local clustering is the dominant cost factor in MAD-C's completion time. Therefore, we expect that MAD-C can be modified into an even faster method if faster local clustering techniques (e.g., parallelization using GPUs) are employed. Moreover, we discussed how to adopt MAD-C's summaries in order to efficiently identify objects in certain areas of the environment for geofencing purposes. Other potential uses of the approach can include synergies of MAD-C with spatio-temporal monitoring applications that involve distributed clustering and connectivity, e.g. [100–103] or other methods targeting continuous, efficient, approximate 2-tier data analysis, e.g. [28].

Chapter 3

Parallel Approximate Clustering and Applications

PARMA-CC: A Family of Parallel Multiphase Approximate Cluster Combining Algorithms

Amir Keramatian, Vincenzo Gulisano, Marina Papatriantafilou and Philippas Tsigas

This Chapter is an adaptation of the article that is under review after minor revision in the *Journal of Parallel and Distributed Computing (JPDC)*.

A preliminary version of this article appeared in:
The proceedings of the 21st International Conference on Distributed Computing and Networking (ICDCN), pp. 20:1-20:10. ACM, 2020.

Summary

Clustering is a common task in data analysis applications. Despite the extensive literature, the continuously increasing volumes of data produced by sensors (e.g., rates of several MB/s by 3D scanners such as LIDAR sensors), and the time-sensitivity of the applications leveraging the clustering outcomes (e.g., detecting critical situations such as detecting boundary crossing from a robot arm that could injure human beings) demand for efficient data clustering algorithms that can effectively utilize the increasing computational capacities of modern hardware. To that end, we leverage approximation and parallelization, where the former is to scale down the amount of data, and the latter is to scale up the computation. Regarding parallelization, we explore a design space for synchronization and workload distribution among the threads. As we study different parts of the design space, we propose representative Parallel Multiphase Approximate Cluster Combining, abbreviated as PARMA-CC, algorithms.

We show that PARMA-CC algorithms yield equivalent clustering outcomes despite their different approaches. Furthermore, we show that certain PARMA-CC algorithms can achieve higher efficiency with respect to certain properties of the data to be clustered. Generally speaking, in PARMA-CC algorithms, parallel threads compute summaries associated with clusters of data (sub)sets. As the threads concurrently combine the summaries, they construct a comprehensive summary of the sets of clusters. By approximating a cluster with its respective geometrical summaries, PARMA-CC algorithms scale well with increased data volumes, and, by computing and efficiently combining the summaries in parallel, they enable latency improvements. PARMA-CC algorithms utilize special data structures that enable parallelism through in-place data processing. As we show in our analysis and evaluation, PARMA-CC algorithms can complement and outperform well-established methods, with significantly better scalability, while still providing highly accurate results in a variety of data sets, even with skewed data distributions, which cause the traditional approaches to exhibit their worst-case behaviour.

3.1 Introduction

Data clustering, the task of grouping data points into sets of close-by points, is a research thread active since decades. Among many applications and use-cases, clustering algorithms are utilized in safety and management applications that monitor environments to (i) detect areas with high space contention and support decisions to e.g., minimize hazards, plan road networks or schedule transport systems, and (ii) identify objects (e.g., a self-driving vehicle) exhibiting dangerous or critical behavior (e.g., crossing a geofence or on a collision course with an obstacle). Despite the large body of work on data clustering (e.g., [2, Ch. 11-16] and references therein), deploying such applications, critical in Internet-of-Things- (IoT-) based systems, remain challenging due to requirements such as:

- handling large data volumes (for example geolocation data gathered by numerous GPS (Global Positioning System) devices over a period of time and/or readings from LIDAR (Light Detection and Ranging) sensors which scan their surroundings via rotating arrays shooting laser beams, producing several MB/s of *point cloud* data),
- time constraints on data processing,
- efficient data processing for a wide range of data properties.

A parallel approach utilizing approximation can open up possibilities to appropriately address the above issues. Approximation reduces the required amount of workload at the expense of ideally small, controllable reduction of accuracy. For example, a recent work proposing MAD-C (Multi-stage Approximate Distributed Cluster-combining) [104] provides evidence regarding the advantages of approximation. MAD-C, being a distributed algorithm for approximating the Euclidean clustering algorithm, multiplicatively reduces the computational workload through approximation at the cost of marginal reduction in the clustering accuracy.

MAD-C's approximation approach aligns with the first part of the "scale down, scale up, scale out" message, summarized by Gibbons in [24], and paves the way to consider the second part, which is about proper utilization of parallelism, already omnipresent in contemporary computing architectures at all levels. To tackle this issue, in this work we address questions regarding the following: Can shared memory boost scalability (i.e. have improved time efficiency with increased number of threads)? Can work-partitioning, scalability and high-degree of accuracy co-exist? How can approximation contribute to super-linear scalability in the number of threads on multi-core systems and maintain high degree of accuracy? Furthermore, can adjusting the algorithm according to the data properties improve scalability? Moreover, as IoT applications leverage numerous types of data with variety of different properties, can the latter affect how much the available computational capacity is utilized by an algorithm? These questions are not jointly answered in the literature (cf. also § 3.10).

To answer the aforementioned questions, we propose a family of Parallel Multiphase Approximate Cluster Combining methods (PARMA-CC). The PARMA-CC algorithms are designed to achieve high scalability over a spectrum

of different properties of data. We show how to utilize the shared memory in a way that supports parallel execution of threads sharing the workload. Furthermore, we show how synchronization among the threads is performed to achieve high scalability over a spectrum of data properties. Because of our novel data structures and their algorithmic implementations, several operations require nearly constant time and enable incremental, in-place processing, gradually constructing the final result by connecting pieces of the data structure. We analyse the properties of the algorithms in the PARMA-CC family, and we show they all achieve equivalent clustering results. Furthermore, we study the efficiency, scalability and accuracy of the PARMA-CC algorithms, also complementing and comparing with well-established methods such as Euclidean clustering algorithm [5], DBSCAN [47], and PDS-DBSCAN [63]. We supplement the analysis with a detailed experimental study, using both LIDAR and GPS data sets. Our results show efficiency in scaling and in preserving accuracy, even with high numbers of threads and large data sets (that can be challenging for existing clustering algorithms) and give practical evidence for the results in the analysis and the benefits of the different approaches for different properties and correlations of the data features.

The remainder of this chapter is organized as follows. In § 3.2, we discuss the preliminaries. We outline the design ideas behind PARMA-CC algorithms in § 3.3. We propose the algorithmic description of the PARMA-CC algorithms in § 3.4 and § 3.5. In § 3.6, we describe our proposed data structures and their algorithmic implementations. In § 3.8, we present a discussion regarding the trade-offs among the PARMA-CC algorithms, further use cases, and some generalizations. We theoretically analyze the PARMA-CC algorithms in § 3.7. We present our empirical evaluation in § 3.9. We discuss the related work and conclusions in § 3.10 and § 3.11, respectively.

3.2 Preliminaries

3.2.1 System Model and Problem Description

We consider a multi-core shared-memory system supporting parallel executions of `K` threads, denoted $\mathtt{t}_1$, $\mathtt{t}_2$, $\cdots$, $\mathtt{t}_K$. Threads access data via `read`, `write` and `read-modify-write` atomic operations. We utilize `CAS`[1] (abbreviating compare-and-swap) and `FAA`[2] (abbreviating fetch-and-add), two commonly used `read-modify-write` atomic operations, supported by all contemporary general purpose processors.

Input Data: `D` denotes the input dataset, a set of `N` points/observations, where each observation contains one or more real-valued features in a metric space (i.e., each feature corresponds to a dimension in the input space), over which distances between points can be calculated. For instance, `D` can be a *point cloud*, i.e., a set of measurements in the 3D space, gathered by one or more LIDAR sensors, or it can contain geolocation data gathered by several GPS

[1] `CAS(var, oldVal, newVal)` atomically changes the value stored at `var` to `newVal` if the value stored at `var` is `oldVal` and returns "true" in such a case, else it does not take any effect and returns "false".

[2] `FAA(var, delta)` atomically adds value `delta` to the value stored at variable `var` and returns the value of the variable.

trackers over a period of time. It is worth noting that a LIDAR sensor gathers a point cloud by targeting laser beams and measuring the time for the laser beams to get reflected back to the sensor. Furthermore, the sensor typically rotates to give a 360° view [105]. Therefore, a point cloud gathered by such a sensor is *angularly sorted* in time.

Problem Description: Given an input dataset `D`, the goal is to partition `D` into an unknown number of mutually disjoint sets (a.k.a clusters) where the points inside each cluster satisfy some predetermined distance-based or density-based criteria. To that end, we aim for an efficient, scalable parallel approximate solution to assign a clustering label to each point in `D` according to the cluster to which the point belongs. The approximation, used to reduce calculations regarding the enforcement of the distance or density criteria, must have high accuracy. As an end result, each cluster should be characterized by its point set (i.e., the cluster members) and also a *volumetric representation* of the cluster.

Objectives: To solve the aforementioned problem, we aim for a set of highly parallel, time-efficient, and scalable algorithms tailored for different data properties in order to properly utilize the available computational power. Regarding guarantees in presence of concurrency, a common consistency goal is that for every parallel execution, there exists a sequential execution that produces an equivalent result. Furthermore, the algorithms must be able to combine efficiency and accuracy benefits. Regarding efficiency, the evaluation metrics are *completion time* and *scalability*, where the scalability of a concurrent algorithm can be measured as the ratio of its completion time when running with one thread, over its completion time when running with `K` threads. The accuracy is measured with respect to the results of an exact baseline method using *rand index* [51, 106], a commonly used measure of clustering similarity. Given two clusterings of the same set, *rand index* measures the ratio of the number of pairs of elements that are either clustered together or separately in both clusterings, to the total number of pairs of elements.

3.2.2 Background

For several of the technical parts of the algorithm descriptions, the following algorithmic and concurrency-related terms are useful to introduce here: A concurrent algorithm is *wait-free* if all the threads can make progress independently of each other. A concurrent implementation of a data object is *linearizable* if the effects of concurrent operations appear instantaneously and are consistent with the sequential specification of the object [107]. An operation implementation is *in-place* if it directly modifies parts of a data structure without making new copies of the latter.

We consider *distance-based* and *density-based* clustering. The points in a distance-based cluster satisfy some minimum distance criteria, and the points in density-based clusters form contiguous region of high-density, separated by contiguous low-density ones. We review PCL-EC (the Point-Cloud-Library's Euclidean clustering algorithm) [5] as a representative of a distance-based clustering. Representing density-based clustering, we cover DBSCAN [47] (Density-Based Spatial Clustering of Applications with Noise) and an established parallel variant, PDSDBSCAN [63]. We refer to PCL-EC and to DBSCAN as exact sequential distance-based and density-based baselines, respectively.

PCL-EC partitions a data set into an a priori unknown number of clusters, so that each cluster has at least `minPts` points, and within each cluster, each point lies in ϵ-radius neighbourhood of at least another point in the same cluster, for parameters `minPts`, ϵ. Non-clustered points are identified as *noise*. Using kd-trees for efficient neighbourhood search, PCL-EC's expected and worst-case time complexities are respectively $\mathcal{O}(\mathtt{N} \log \mathtt{N})$ and $\mathcal{O}(\mathtt{N}^2)$, see [5, Ch. 4].

DBSCAN partitions a data set into an a priori unknown number of clusters such that a cluster consists of at least one *core point* and all the points that are *density-reachable* from it. Point p is a core point if it has at least `minPts` points in its ϵ-radius neighbourhood. Point q is *directly reachable* from p if q lies in the ϵ-radius neighbourhood of p. Point q is density-reachable from p, if q is directly reachable either from p or another core point that is density-reachable from p. Non-core points that are not density-reachable from any core-points are outliers [66]. The expected and worst-case time complexities of DBSCAN are respectively $\mathcal{O}(\mathtt{N} \log \mathtt{N})$ and $\mathcal{O}(\mathtt{N}^2)$ [20].

PDSDBSCAN [63] is a parallel version of DBSCAN. It parallelizes the work through partitioning the points and merging partial clusters consisting of points, maintained via a *disjoint-set* data structure, that facilitates maintaining a collection of disjoint sets supporting in-place *union* and *find* operations [108, Ch. 21.1]

3.3 The PARMA-CC Family of Algorithms

Clustering is a global aggregate function, and as such it is far from being an embarrassingly-parallel application; hence, concurrency (parallel tasks working on subsets of data) and synchronization (putting together the results of the data subsets) imply natural trade-offs. We propose the PARMA-CC algorithms, abbreviating Parallel Multiphase Approximate Cluster Combining, to explore the design space for parallelism, in conjunction with appropriately designed data structures, to provide alternative options for different scenarios.

For the exposition of the algorithms, we consider LIDAR data as example, as it enables more intuitive descriptions. Nonetheless, the algorithms can process various types of data, and we evaluate them with LIDAR and GPS data.

3.3.1 High-level View

On one side of the design space, the algorithms in the family target a *coarse-grained synchronization* approach through which operations on disjoint elements are performed in a data parallel fashion, but operations on the shared elements are performed in a mutually exclusive manner. On the other side of the design space, the algorithms target a *fine-grained synchronization* approach through which operations are performed in a fully concurrent fashion in a wait-free manner. The coarse-grained synchronization approach utilizes a scheme for data access control that can take advantage of a work-saving mechanism while the fine-grained synchronization approach eliminates the inherent waiting that is present in the more coarse-grained synchronization one. Furthermore, based on an orthogonal aspect, the algorithms in the family leverage either *coarse-grained*

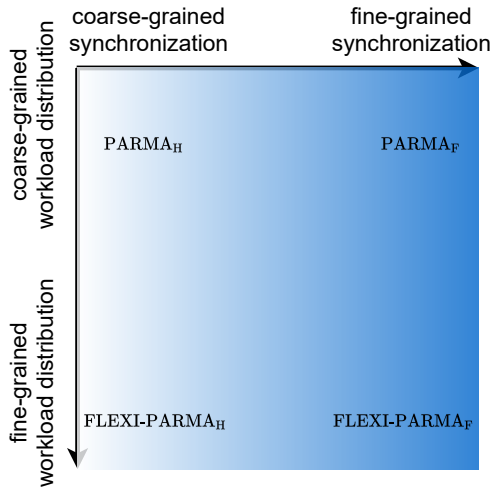


Figure 3.1: The design space of PARMA-CC algorithms.

or *fine-grained workload distribution*. Figure 3.1 visualizes the aforementioned aspects of the design space.

For i $\in \{1, \cdots, \mathtt{S}\}$, ptCloud$_i$s denote mutually disjoint subsets of `D`, where each ptCloud$_i$ is a *split* of `D`, and `S` is the *number of splits*. Each ptCloud$_i$ can be, e.g., the i-th chunk of `N/S` consecutive points in `D`. Figure 3.2a shows a hypothetical dataset `D` being split into four splits. In a PARMA-CC algorithm, `K` threads in parallel cluster the splits and summarize the locally detected clusters. Afterwards, the threads combine the local summaries to create a holistic summary. Lastly, according to the combined summary, points in `D` are relabeled. Alg. 3.1 shows the high-level description of a PARMA-CC algorithm. We will see how each PARMA-CC algorithm is designed based on its position in the design space in Figure 3.1. Furthermore, we will see that all the PARMA-CC algorithms yield equivalent clustering results.

Algorithm 3.1 Outline of the three phases of a PARMA-CC algorithm

```
1:  let K be the number of CPU threads
2:  let ptCloud_1, ···, ptCloud_S be splits of D
3:  let 𝔽 be an appropriately designed shared data structure
4:  for all K threads in parallel do
5:    phase I:
6:      while ∃ ptCloud_i to be clustered do
7:        cluster ptCloud_i and summarize its local clusters in 𝔽
8:        index the summaries in split-summary φ_i
9:        announce the creation of φ_i
10:   phase II:
11:     create objects by detecting and grouping matching summaries in 𝔽
12:   phase III (starts when all threads have reached here):
13:     while ∃ ptCloud_i to be relabeled do
14:       relabel the points in ptCloud_i according to the combined results
```

Challenges

The high-level design in Alg. 3.1 implies challenges regarding data structures, workload distribution, and synchronization, outlined in the following:
Data Structures: The design and algorithmic implementation of the shared data structure in Alg. 3.1 (denoted by $\mathbb{F}$) must be carried out with regard to concurrent updates by several threads as well as in-place operations to further facilitate scaling-up through shared memory parallelism.
Workload Distribution: The workload distribution and *work-partitioning* mechanism among the threads in a PARMA-CC algorithm must facilitate effective collaboration with minimal synchronization and contention overheads.
Synchronization: The synchronization and communication among the threads in all PARMA-CC algorithms must ensure consistency (in the final outcome) despite the diverse algorithmic choices suggested in the design space in Figure 3.1.

3.3.2 Rudiments and Definitions

Here we provide general details relevant to all PARMA-CC algorithms. For better intuition, we use a summarization technique utilizing *bounding ellipsoid*s for the exposition of the methods. However, we will see in § 3.8 that the choice of the summarization technique is orthogonal to the behaviour of PARMA-CC algorithms.

Definition 3.1. *[Objects, Split-summaries, Maps]*

- *A* local cluster *is a cluster of points identified by a clustering algorithm (e.g., DBSCAN or PCL-EC) performed on a split of the input data. A* bounding ellipsoid *is a volumetric summary of a local cluster.*
- *A pair of ellipsoids* $\langle \mathtt{e}_1, \mathtt{e}_2 \rangle$ *can* overlap directly *or* indirectly*;* $\mathtt{e}_1$ *and* $\mathtt{e}_2$ *directly overlap if* $\mathtt{e}_1$ *and* $\mathtt{e}_2$ *geometrically overlap;* $\mathtt{e}_1$ *and* $\mathtt{e}_2$ *indirectly overlap if there is an ellipsoid* $\mathtt{e}'$ *such that both pairs* $\langle \mathtt{e}_1, \mathtt{e}' \rangle$ *and* $\langle \mathtt{e}_2, \mathtt{e}' \rangle$ *overlap, either directly or indirectly.*
- *A* split-summary φ_i *is a set of ellipsoids corresponding to the detected clusters in the i-th split. Figure 3.2b shows the split-summaries corresponding to the data splits in Figure 3.2a.*
- *An* object *consists of a set of mutually overlapping ellipsoids. Given an ellipsoid* $\mathtt{e}$*, let* $\mathbb{O}_{\mathtt{e}}$ *denote the object in which* $\mathtt{e}$ *belongs.*
- *Two objects* overlap *if there is at least a pair of overlapping ellipsoids (one in each object). Two overlapping objects can get* merged*, forming a bigger object containing all the ellipsoids in the original objects.*
- *A* map *is a set of objects. Figure 3.3 shows several maps.*

At the heart of each PARMA-CC algorithm lies a shared data structure called the *ellipsoid forest*, denoted by $\mathbb{F}$ in Alg. 3.1. An ellipsoid forest enables *multi-threaded in-place processing and access to ellipsoids, supporting efficient indexing and retrieval of objects in maps and ellipsoids in objects and split-summaries.* At the end of phase I, each ellipsoid, summarizing a local cluster,

becomes a *singleton* in the ellipsoid forest upon creation. As the forest evolves in phase II, overlapping ellipsoids get grouped together, i.e., by forming objects.

Definition 3.2. *[Ellipsoid Forest - Extended Disjoint Set Data Structure] We propose to implement the ellipsoid forest as an* extended disjoint-set data structure *[108, Ch. 21], i.e., a data structure that can store disjoint sets of ellipsoids, representing growing* objects. *Here, in a disjoint-set, a* tree *represents an object, and the root of a given tree is called the* representative *of the associated object. Similar to a disjoint-set, an ellipsoid forest supports the following operations: (i)* `findRoot` *returns the representative of the object containing a given ellipsoid, and (ii)* `merge` *replaces two given objects with their union.*
We propose two extensions of the disjoint-set data structure, resulting in two variants of an ellipsoid forest data structure, in particular through the following:

- *operation* `mapCombining`*, which, given maps* $\mathbb{M}_i$ *and* $\mathbb{M}_j$*, for each* $\mathbb{O}$ *in* $\mathbb{M}_i$ *and* $\mathbb{O}'$ *in* $\mathbb{M}_j$*, merges* $\mathbb{O}$ *and* $\mathbb{O}'$ *if they overlap, and it returns a new map that indexes the resulting objects (merged and not merged objects of* $\mathbb{M}_i$ *and* $\mathbb{M}_j$*). The operation is to be invoked in a synchronized, hierarchical order, to produce a final map by combining evolving partial maps, and hence we name the extended data structure* hierarchical ellipsoid forest*; or*

- *operation* `ellipsoidLinking`*, which, given split-summaries* φ_i *and* φ_j*, for each pair of ellipsoids* $\mathtt{e}$ *and* $\mathtt{e}'$ *in* φ_i *and* φ_j*, if they overlap, merges the objects they are part of, i.e.,* $\mathbb{O}_{\mathtt{e}}$ *and* $\mathbb{O}_{\mathtt{e}'}$*. The operation does not return any index, it only updates internal links in the composite data structure. It can be invoked concurrently in an asynchronous fashion, to perform linking between all pairs of split-summaries, and hence we name the extended data structure* flat ellipsoid forest*.*

Table 3.1 summarizes the ellipsoid forests' extended API.

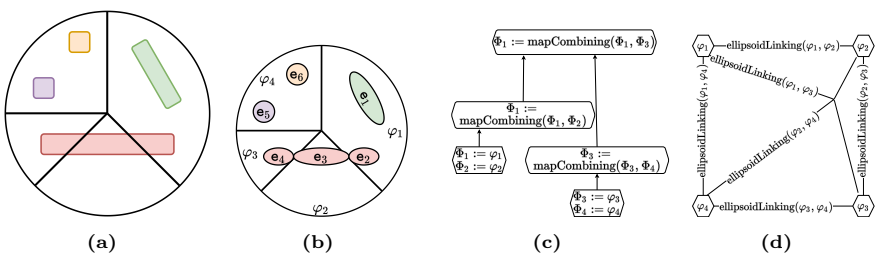


Figure 3.2: *(a) Dataset split into four splits. (b) Split summaries of local clusters. (c) Hierarchy* $\mathbb{H}$ *of* `mapCombining` *operations in a hierarchical PARMA-CC algorithm. (d)* `ellipsoidLinking` *operations in a flat PARMA-CC algorithm.*

The Three Phases of a PARMA-CC Algorithm

Having introduced the concept of the ellipsoid forest, we give a refined outline of a PARMA-CC-family algorithm.

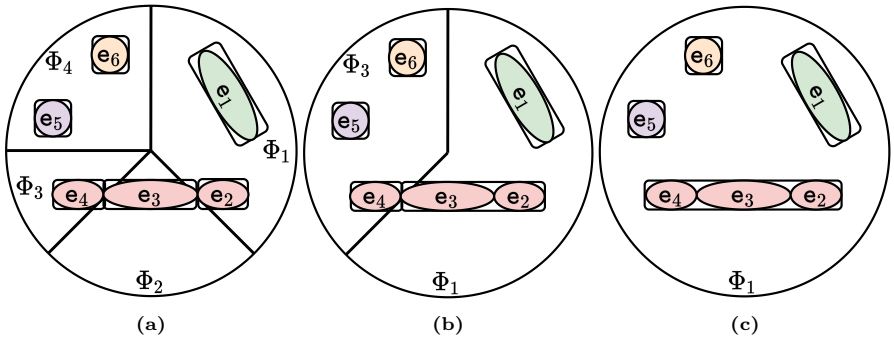


Figure 3.3: *Maps in a hierarchical PARMA-CC algorithm. Delimiting boxes indicate detected objects in each map. (a) Initial contents of the maps. (b) Maps* $\mathbb{M}_1$ *and* $\mathbb{M}_3$ *after operations* $\mathbb{M}_1$*:=*`mapCombining`*(*$\mathbb{M}_1$*,*$\mathbb{M}_2$*)* *and* $\mathbb{M}_3$*:=*`mapCombining`*(*$\mathbb{M}_3$*,*$\mathbb{M}_4$*), respectively. (c)* $\mathbb{M}_1$ *after operation* $\mathbb{M}_1$*:=*`mapCombining`*(*$\mathbb{M}_1$*,*$\mathbb{M}_3$*).*

Phase I: The goal here is to efficiently organize volumetric summaries of local clusters in the shared memory, facilitating efficient operations in phase II. To that end, the threads collaboratively cluster the data splits and create the split-summaries φ_is. The aforementioned steps are outlined in Alg. 3.1 l.6-9.

Phase II: The objective in this phase, outlined in Alg. 3.1 l. 11, is to concurrently detect and group overlapping ellipsoids in the ellipsoid forest in a scalable manner.

Phase III: This phase's objective is to assign clustering labels to points in `D` such that all points that relate to the same object are given the same label, different from labels of points that belong to other objects. Therefore, to relabel the points associated with the ellipsoids in a certain object, we use the identity of the root of the associated object in the ellipsoid forest, retrieved by `findRoot`. To make sure that the objects in the ellipsoid forest do not change any more when this phase is executed, a thread should start its phase III only after all threads have finished their phase II. The aforementioned steps are outlined in Alg. 3.1 l.13-14.

The algorithmic details of phases I-III are determined based on the choices in the design space of PARMA-CC algorithms. Particularly, the algorithmic details of phases I and III are determined based on the workload distribution aspect of the design space, and the algorithmic details of phase II are determined based on the synchronization aspect of the design space. We elaborate on the two aspects of the design space in the following subsection.

3.3.3 The Design Space of PARMA-CC Algorithms

We explained in § 3.3.1 that PARMA-CC algorithms cover a design space concerning two orthogonal aspects: (i) synchronization, and (ii) workload distribution.

Synchronization via Hierarchical Ellipsoid Forest vs Synchronization via Flat Ellipsoid Forest

The synchronization aspect of the design space mainly concerns the manner in which overlapping ellipsoids are detected and grouped together. As ellipsoids are stored in the ellipsoid forest, the key element regarding this aspect is the ellipsoid forest. As noted in Definition 3.2, there are two forest types, named hierarchical and flat. From now on, a hierarchical PARMA-CC algorithm is one that utilizes the hierarchical forest, and a flat PARMA-CC algorithm is one that utilizes the flat forest.

Table 3.1: Ellipsoid Forest's Extended API (the algorithmic implementations are presented in § 3.6)

operation	forest type	description
`mapCombining`($\mathbb{M}_i$, $\mathbb{M}_j$)	hierarchical	for each $\mathbb{O}$ in $\mathbb{M}_i$ and each $\mathbb{O}'$ in $\mathbb{M}_j$, merges $\mathbb{O}$ and $\mathbb{O}'$ if $\mathbb{O}$ and $\mathbb{O}'$ overlap, returns the combined map.
`ellipsoidLinking`(φ_i, φ_j)	flat	for each $\mathtt{e}$ in φ_i and each $\mathtt{e}'$ in φ_j, merges objects $\mathbb{O}_\mathtt{e}$ and $\mathbb{O}_{\mathtt{e}'}$ if $\mathbb{O}_\mathtt{e}$ and $\mathbb{O}_{\mathtt{e}'}$ overlap.

In a hierarchical PARMA-CC algorithm, the order of performing operations is synchronized via a tree $\mathbb{H}$, spanning over nodes that represent maps in the process of constructing the overall outcome (cf. Definition 3.1). Particularly, every node in $\mathbb{H}$ instructs performing `mapCombining` on two maps. Operations in disjoint branches of $\mathbb{H}$ can be performed concurrently, but in the same branch, the order of performing the operations must follow the hierarchy, starting from the leaves and continuing upwards. Figure 3.2c shows a hierarchy applicable on the maps in Figure 3.3a, i.e., the maps initiated by the split-summaries in Figure 3.2b. Accordingly, Figure 3.3b shows the contents of maps $\mathbb{M}_1$ and $\mathbb{M}_3$ after operations $\mathbb{M}_1$:= `mapCombining`($\mathbb{M}_1$,$\mathbb{M}_2$) and $\mathbb{M}_3$:= `mapCombining`($\mathbb{M}_3$,$\mathbb{M}_4$), respectively. Finally, Figure 3.3c shows the content of $\mathbb{M}_1$ after operation $\mathbb{M}_1$:= `mapCombining`($\mathbb{M}_1$,$\mathbb{M}_3$). In Figure 3.3, the objects in each map are distinguished by delimiting boxes.

In a flat PARMA-CC algorithm, there is no need to synchronize the order of performing operations because `ellipsoidLinking` utilizes asynchronous concurrent linearizable operations (similar to [109]) which makes it possible to perform consistently the `ellipsoidLinking` operations (which are commutative and associative, as we show in the more detailed description sections) in a fully concurrent fashion on all pairs of split-summaries. Figure 3.2d outlines the `ellipsoidLinking` operations corresponding to the split-summaries in Figure 3.2b. Note that performing the `ellipsoidLinking` operations in Figure 3.2d will result in detecting the same objects shown in Figure 3.3c.

Basic Workload Distribution vs Flexible Workload Distribution

Another key aspect of the design space (as we outlined in the beginning of the section and in Figure 3.1), is the distribution of the local clustering and local relabeling tasks (i.e., phase I and phase III workload) among the threads. To that end, PARMA-CC algorithms are categorized into two groups. In the first group, which we refer to as *basic*, the workload in phase I and phase III are distributed among the threads in a *work-sharing* [110] style by statically

assigning each task to a processor. In the second group, which we refer to as *flexible*, the workload is partitioned into a large number of tasks (larger than the number of threads in the system) available as a *shared pool*, from which the threads compete to take tasks in a *work-stealing* [110] fashion. From now on, a *basic PARMA-CC* algorithm is one that utilizes the basic workload distribution approach, and a *flexi PARMA-CC* is one that utilizes the flexible workload distribution approach.

Basic Workload Distribution: The basic workload distribution assumes a one-to-one relation between the number of threads (`K`) and the number of splits (`S`). Concretely, for each $i \in \{1, \cdots, \texttt{K}\}$, the local clustering and relabeling tasks associated with ptCloud_i are performed statically assigned to thread $\texttt{t}_\texttt{i}$. We cover the basic PARMA-CC algorithms in § 3.4.

Flexible Workload Distribution: Notice that in a basic workload distribution, the duration of performing the local clustering tasks can vary between splits even when they contain the same number of points and the threads are equally fast. The latter is due to the fact that the local clustering algorithm employs a spatial data structure for indexing the points. Consequently, with different distributions of points in each split, the associated cost of using the data structure varies. Hence, the threads that finish their local clustering task earlier will have to wait in subsequent phases of the algorithm. To alleviate the aforementioned problem, a flexi PARMA-CC algorithm breaks down the local clustering and local relabeling tasks into fine-grained chunks by allowing `S` to be larger than `K`. A flexi PARMA-CC algorithm accommodates a shared pool of local clustering and relabeling tasks which can be *booked* by each thread in a wait-free manner. We cover the flexi PARMA-CC algorithms in § 3.5.

Observation 3.1. *In the special case where* `S` *is equal to one, PARMA-CC algorithms and the exact sequential baseline (see § 3.2.2) are identical. Therefore, the scalability of a PARMA-CC algorithm (measured as the ratio of its completion times with one thread and with* `K` *threads) equals the ratio of the completion time of the exact sequential baseline to the completion time of the PARMA-CC algorithm.*

3.4 Basic Members of the PARMA-CC Family

Hierarchical Parallel Multiphase Approximate Cluster Combining, abbreviated as $\texttt{PARMA}_\texttt{H}$, and Flat Parallel Multiphase Approximate Cluster Combining, abbreviated as $\texttt{PARMA}_\texttt{F}$, are the two basic members of the PARMA-CC family. We introduce $\texttt{PARMA}_\texttt{H}$ in § 3.4.1 and $\texttt{PARMA}_\texttt{F}$ in § 3.4.2. In § 3.5, we discuss how $\texttt{PARMA}_\texttt{H}$ and $\texttt{PARMA}_\texttt{F}$ serve as a basis for the flexi members of the family[3].

3.4.1 $\texttt{PARMA}_\texttt{H}$

$\texttt{PARMA}_\texttt{H}$ is a basic PARMA-CC algorithm that utilizes the hierarchical ellipsoid forest (see § 3.3.3). In $\texttt{PARMA}_\texttt{H}$, each `mapCombining` in hierarchy $\mathbb{H}$ is uniquely associated with a thread which is responsible for performing the associated

[3]The gray lines in the pseudocodes indicate parts that have been described in previous sections and are marked in this way to facilitate the focus of the different parts of each algorithm.

Algorithm 3.2 $\texttt{PARMA}_{\texttt{H}}$ algorithm

```
1:  let ℍ be a combine hierarchy
2:  let each mapCombining in ℍ be uniquely associated with a thread
3:  let S:=K
4:  for all thread t_i | i ∈ {1,··· ,K} in parallel do
5:    phase I:
6:      cluster ptCloud_i & summarize its local clusters
7:      index the summaries in φ_i
8:      𝕄_i:= φ_i
9:      signal the responsible thread on the first level of ℍ that 𝕄_i is ready
10:   phase II:
11:     if t_i is responsible for mapCombining(𝕄_m,𝕄_n) then
12:       wait to receive signals that 𝕄_m and 𝕄_n are ready
13:       𝕄_m:= mapCombining(𝕄_m, 𝕄_n)
14:       signal the responsible thread in the next level of ℍ (if any) that 𝕄_m is ready
15:   phase III (starts when all threads have reached here):
16:     relabel the points in ptCloud_i based on their objects
```

mapCombining. To make sure that the content of the maps are finalized before a thread performs its mapCombining, it must wait until it receives *signal*s that the associated maps are *ready*. Alg. 3.2 gives a high-level view of $\texttt{PARMA}_{\texttt{H}}$. We study phase I and phase II of $\texttt{PARMA}_{\texttt{H}}$ in the following. We avoid repeating phase III of $\texttt{PARMA}_{\texttt{H}}$ because it is identical to the provided details of the corresponding phase in § 3.3.2.

Phase I: After having created the split-summary $\varphi_{\texttt{i}}$, thread $\texttt{t}_{\texttt{i}}$ initializes map $\mathbb{M}_i$ by the content of $\varphi_{\texttt{i}}$, as indicated in Alg. 3.2 l.8. Afterwards, $\texttt{t}_{\texttt{i}}$ signals the responsible thread on the first level of $\mathbb{H}$ that $\mathbb{M}_i$ is ready.

Phase II: Assuming $\texttt{t}_{\texttt{i}}$ is responsible for mapCombining($\mathbb{M}_m$, $\mathbb{M}_n$), after having recieved the signals that $\mathbb{M}_m$ and $\mathbb{M}_n$ are ready, $\texttt{t}_{\texttt{i}}$ performs the associated mapCombining and updates $\mathbb{M}_m$, as indicated in Alg. 3.2 l.13. Then, $\texttt{t}_{\texttt{i}}$ signals the responsible thread in the next level of $\mathbb{H}$ (if any) that $\mathbb{M}_m$ is ready, as shown in Alg. 3.2 l.14.

3.4.2 $\texttt{PARMA}_{\texttt{F}}$

$\texttt{PARMA}_{\texttt{F}}$ is a basic PARMA-CC algorithm that utilizes the flat ellipsoid forest (see § 3.3.3). In $\texttt{PARMA}_{\texttt{F}}$ the elements in the ellipsoid forest are accessed and modified in a fully concurrent manner, i.e. no ordering is required. The latter holds because $\texttt{PARMA}_{\texttt{F}}$ utilizes the ellipsoidLinking operations to detect and group the overlapping ellipsoids. We will cover the algorithmic implementation of the data structure associated with ellipsoidLinking and their consistency guarantees in the presence of concurrent operations in § 3.6.3.

In $\texttt{PARMA}_{\texttt{F}}$, the ellipsoidLinking tasks are distributed among the threads based on the availability of tasks and the availability of unoccupied threads. To that end, each thread, after having performed the local clustering task and having created the associated split-summary, should reveal the availability of the new split-summary and the associated ellipsoidLinking tasks to the rest of the threads, so the unoccupied threads can perform the associated tasks. We propose to utilize an array V, for storing the *status* of the ellipsoidLinking tasks:

Algorithm 3.3 PARMA$_{\mathtt{F}}$ algorithm

```
1:  let V be {v_{m,n} | m,n ∈ {1,2,···,S}, m < n} (see Definition 3.3)
2:  let S:=K
3:  for all thread t_i | i ∈ {1,···,K} do
4:    phase I:
5:      cluster ptCloud_i & summarize its local clusters
6:      index the summaries in φ_i
7:      for all v ∈ V|v = v_{i,x} or v = v_{x,i} do
8:        atomically increment v //e.g., using FAA
9:    phase II:
10:     while ∃(m,n) | corresponding task to v_{m,n} not booked do
11:       if corresponding task to v_{m,n} is booked // (e.g. using CAS(v_{m,n}, 2, 3)) then
12:         ellipsoidLinking(φ_m,φ_n)
13:   phase III (starts when all threads have reached here):
14:     relabel the points in ptCloud_i based on their objects
```

Definition 3.3. *Let* V *be a set of* status *values, each one associated with an* ellipsoidLinking *task on a certain pair of maps. As* ellipsoidLinking *is symmetric,* V *is defined as* $\{v_{m,n} | m,n \in \{1,2,\cdots,\mathtt{S}\}, m < n\}$*, where* $v_{i,j}$ *indicates the status of* ellipsoidLinking(φ_m, φ_n) *and can have any of the following values:* 0*: when neither* φ_m *nor* φ_n *is created (initial value);* 1*: when one of* φ_m *or* φ_n *is ready;* 2*: when both* φ_m *and* φ_n *are ready, but* ellipsoidLinking(φ_m, φ_n) *is not yet booked,* 3*: when* ellipsoidLinking(φ_m, φ_n) *is booked (to be performed by the thread that booked it).*

To make sure concurrent updates on V are performed correctly, the threads use *atomic synchronization primitives* to update the status values.

Alg. 3.3 outlines PARMA$_{\mathtt{F}}$. We study phase I and phase II of PARMA$_{\mathtt{F}}$ in the following. We avoid repeating phase III of PARMA$_{\mathtt{F}}$ because it is identical to the provided details of the corresponding phase in § 3.3.2.

Phase I: In this phase, after having created split-summary $\varphi_{\mathtt{i}}$, thread $\mathtt{t}_{\mathtt{i}}$ updates the status values of the affected ellipsoidLinking tasks. To that end, it atomically increments the status values of each affected task (e.g., by performing FAA), as shown in Alg. 3.3 l.7-8.

Phase II: A thread in this phase keeps iterating through the status values in V. As the thread finds a task which is not booked yet, it tries to atomically *book* the task (e.g., via CAS operation to change the its status from two to three). Upon successful booking, the thread performs the respective task. The aforementioned steps are shown in Alg. 3.3 l.10-l.12.

3.5 Flexi Members of the PARMA-CC Family

A flexi PARMA-CC targets *flexible workload distribution* among the threads, in particular dividing the local clustering and local relabeling tasks into fine-grained chunks, through allowing S to be larger than K. Following the discussion in § 3.3.3, the flexible workload distribution decreases the potential amount of the threads' waiting time; therefore, it increases the utilization of resources. We introduce FLEXI-PARMA$_{\mathtt{H}}$, and FLEXI-PARMA$_{\mathtt{F}}$, flexi versions of PARMA$_{\mathtt{H}}$ and PARMA$_{\mathtt{F}}$, respectively.

3.5.1 Flexi Shared Phases

As there is not a one-to-one correspondence between the data splits and the threads, we design a wait-free *booking* mechanism for performing the local clustering and local relabeling tasks, which we explain in the following.
Phase I: The goal here is to perform parallel local clustering of `S` splits with `K` threads. Let `LCT`, abbreviating Local Clustering Tasks, be a boolean array of size `S`, where each index shows if the associated local clustering task has been performed. The booking mechanism is similar to the one introduced in § 3.4.2. Phase I is shown in Alg. 3.5 and Alg. 3.4 for the flexi PARMA-CC algorithms. The lines marked by ★ indicate the preparation step for phase II.
Phase III: The goal here is to perform parallel local relabeling of `S` splits with `K` threads. To that end, we utilize a boolean array of size `S` named `LRT`, abbreviating Local Relabeling Tasks, in the same fashion as explained for `LCT`. Phase III in Alg. 3.5 and Alg. 3.4 outline the relabeling steps in flexi PARMA-CC algorithms.

3.5.2 Flexi PARMA-CC Algorithms

We review the uncovered details of Flexi PARMA-CC algorithms in the following.

Algorithm 3.4 `FLEXI-PARMA`$_{\texttt{H}}$ algorithm

```
1:  let LCT and LRT be shared arrays of size S initialized to 0
2:  let Q be a multithreaded queue
3:  let totalNumberOfCombines be initialized to 0
4:  for all K threads in parallel do
5:    phase I:
6:      for splitID ∈ {1, ··· , S} do
7:        if CAS(LCT[splitID], 0, 1) then
8:          cluster ptCloud_splitID & summarize its local clusters
9:          index the summaries in φ_splitID
10:         M_splitID := φ_splitID
11:         Q.push(splitID)(★)
12:   phase II:
13:     while totalNumberOfCombines < S − 1 do
14:       if Q.tryPop(i, j) == success then
15:         M_i := mapCombining(M_i, M_j)
16:         FAA (totalNumberOfCombines, 1)
17:         Q.push(i)
18:   phase III (starts when all threads have reached here):
19:     for splitID ∈ {1, ··· , S}
20:       if CAS(LRT[splitID], 0, 1) then
21:         relabel the points in ptCloud_splitID based on their objects
```

`FLEXI-PARMA`$_{\texttt{H}}$: The goal of this algorithm is to reduce the amount of time that a thread waits for its descendants' maps. To that end, this algorithm utilizes an *agile* mechanism to generate the combine hierarchy $\mathbb{H}$ on the fly. Notably, $\mathbb{H}$ is determined based on the order in which the maps become available. The latter is achieved by utilizing a multithreaded queue `Q`, which holds the indices of the ready maps. As the preparation step (★), for each local clustering task that a thread performs, it inserts the index of the associated map in `Q`. In phase II, a

thread tries to pop two indices from `Q`. If two indices are popped successfully, then it performs `mapCombining` on associated maps, and it will insert the index of the resulting map in `Q`. This process continues until (`S` -1) `mapCombining` operations are performed. To that end, the total number of performed tasks is kept as a global variable that gets incremented atomically. When (`S` -1) tasks are performed, there is only one map index in `Q`, which indexes all the objects in the forest. Alg. 3.4 outlines $\texttt{FLEXI-PARMA}_\texttt{H}$.

$\texttt{FLEXI-PARMA}_\texttt{F}$: This is a flexi version of $\texttt{PARMA}_\texttt{F}$. As the preparation step (★), for each local clustering task that a thread performs, the thread updates the status values of the affected `ellipsoidLinking` tasks, using the technique explained in § 3.4.2. Alg. 3.5 outlines $\texttt{FLEXI-PARMA}_\texttt{F}$.

Algorithm 3.5 $\texttt{FLEXI-PARMA}_\texttt{F}$ algorithm

```
1: let LCT and LRT be shared arrays of size S initialized with 0
2: let V be {v_{i,j} | i, j ∈ {1, 2, ··· , S}, i < j} (see Definition 3.3)
3: for all K threads in parallel do
4:   phase I:
5:     for splitID ∈ {1, ··· , S} do
6:       if CAS(LCT[splitID], 0, 1) then
7:         cluster ptCloud_splitID & summarize its local clusters
8:         index the summaries in φ_splitID
9:         for all v ∈ V | v = v_{splitID,x} or v = v_{x,splitID} do
10:          FAA(v, 1)(★)
11:  phase II:
12:    while ∃(i, j) | v_{i,j} ≠ 3 do
13:      if CAS(v_{i,j}, 2, 3) then
14:        ellipsoidLinking(φ_i, φ_j)
15:  phase III (starts when all threads have reached here):
16:    for splitID ∈ {1, ··· , S}
17:      if CAS(LRT[splitID], 0, 1) then
18:        relabel the points in ptCloud_splitID based on their objects
```

3.6 Ellipsoid Forest Data Structures and Algorithmic Implementation

In this section, we introduce the algorithmic implementation of the ellipsoid forest data structure. We start by introducing the bounding ellipsoid data structure. Afterwards, we study the algorithmic implementation of the hierarchical and flat forests in § 3.6.2 and § 3.6.3, respectively.

3.6.1 The Bounding Ellipsoid Data Structure

In our design, each (bounding) ellipsoid is instantiated in the shared memory and automatically becomes an element in the forest upon creation. The data structure supporting an ellipsoid contains μ and Σ used to represent the ellipsoid's centroid vector and covariance matrix, respectively (see § 3.2.2). Furthermore, it also contains certain fields that are required to maintain the membership of an ellipsoid in the forest. To that end, a unique `ID` is required to identify the ellipsoid in the forest. Furthermore, a *parent* pointer, initialized

to null, is utilized to support the structure of the trees in the forest. Moreover, an ellipsoid requires a *next* pointer and a *rank* value. As we explain in § 3.6.2, the next pointers facilitate efficient enumeration of ellipsoids in objects, and the rank values regulate the heights of the trees resulting from a `mapCombining` operation.

For a given cluster `c`, μ and Σ of the associated bounding ellipsoid are respectively the sample mean and sample covariance of the points in `c`, which can be calculated with $\mathcal{O}(1)$ time complexity. Similarly, the other fields of the bounding ellipsoid data structure can be initialized with $\mathcal{O}(1)$ time complexity. Furthermore, given two ellipsoids, we can determine if they geometrically overlap, using the method described in [83] with $\mathcal{O}(1)$ time complexity.

3.6.2 Hierarchical Ellipsoid Forest

As noted in § 3.3.3, in a hierarchical forest, `mapCombining` operations are performed according to a combine hierarchy $\mathbb{H}$, which regulates the concurrent accesses and operations in the forest. Furthermore, the objects are represented by *enhanced* trees in a hierarchical forest. An enhanced tree facilitates iterating through the ellipsoids in the associated object with constant time per ellipsoid. The latter is achieved by making the ellipsoids in an enhanced tree form a *circular linked-list* via the next pointers (see § 3.6.1).

Compound Operation

Operation `mapCombining`: Given maps $\mathbb{M}_i$ and $\mathbb{M}_j$, for each pair of objects $\langle \mathbb{O}, \mathbb{O}' \rangle$, where $\mathbb{O} \in \mathbb{M}_i$ and $\mathbb{O}' \in \mathbb{M}_j$, `mapCombining` merges $\mathbb{O}$ and $\mathbb{O}'$ if they overlap. Afterwards, the objects in $\mathbb{M}_j$ get linked to $\mathbb{M}_i$. Finally, potential duplicate objects in $\mathbb{M}_i$ are removed. To that end, an unset flag is associated with the root of every object in $\mathbb{M}_i$. Then, every pointer in $\mathbb{M}_i$'s linked-list is iterated: the flag of the associated object's root gets marked if it is not already marked. Otherwise, the corresponding pointer gets removed from $\mathbb{M}_i$'s linked-list because another pointer in $\mathbb{M}_i$ already points to the same object. Alg. 3.6 outlines the algorithmic implementation of `mapCombining`.

Algorithm 3.6 Operation `mapCombining` in a hierarchical ellipsoid forest

```
1:  procedure mapCombining(M_i, M_j)
2:    for O ∈ M_i.list & O' ∈ M_j.list do
3:      if overlap(O, O') then
4:        merge_H(O, O')
5:    M_i.list.pushAll(M_j.list)
6:    unmark all objects in M.list
7:    for O ∈ M.list do
8:      if findRoot_H(O).marked then
9:        M.list.remove(O)
10:     else
11:       findRoot_H(O).marked:= 1
12:   return M_i
```

Enhancements (i) For improved amortized time complexity, we adopt the *path-compression* and *union-by-rank* heuristics [108]; the former flattens the trees, and the latter controls the growth of depth of the trees. To that end, the *rank* value (see § 3.6.1), initially zero, is assigned to each ellipsoid.
(ii) Note that `mapCombining`($\mathbb{M}_i$, $\mathbb{M}_j$) checks all pairs of objects in $\mathbb{M}_i$ and $\mathbb{M}_j$ to merge the overlapping ones. Suppose object $\mathbb{O}$ in $\mathbb{M}_i$ and object $\mathbb{O}'$ in $\mathbb{M}_j$ do not overlap. To determine this, `mapCombining` has to check all pairs of ellipsoids in $\mathbb{O}$ and $\mathbb{O}'$. To avoid this worst-case behaviour, we propose a work-saving *test* that utilizes *delimiting boxes*.

Definition 3.4. *An object's* delimiting box *is the smallest axis-aligned cuboid encapsulating the ellipsoids in the object.*

The delimiting-box test: If the delimiting boxes of objects $\mathbb{O}$ and $\mathbb{O}'$ do not geometrically overlap, then $\mathbb{O}$ and $\mathbb{O}'$ do not overlap, hence effectively saving pairwise checks of the ellipsoids in $\mathbb{O}$ and $\mathbb{O}'$.

Algorithm 3.7 Auxiliary operations in a hierarchical ellipsoid forest

```
1:  procedure overlap(𝕆, 𝕆′)
2:    if ¬overlap(𝕆.dBox, 𝕆′.dBox) then
3:      return false
4:    for e ∈ 𝕆 & e′ ∈ 𝕆′ do
5:      if e and e′ overlap then
6:        return true
7:    return false
8:  procedure merge_H(𝕆, 𝕆′)
9:    e :=findRoot_H(𝕆)
10:   e′ :=findRoot_H(𝕆′)
11:   link_H(e, e′)
12:   swap(e.next, e′.next)
13:   findRoot_H(e).dBox = union(e.dBox, e′.dBox)
14: procedure findRoot_H(e)
15:   if e.parent == ∅ then
16:     return e
17:   else
18:     e.parent:=findRoot_H(e.parent)
19:     return e.parent
20: procedure link_H(e, e′)
21:   if e.rank > e′.rank then
22:     e′.parent := e
23:   else
24:     e.parent := e′
25:     if e.rank == e′.rank then
26:       e′.rank := e′.rank + 1
```

Auxiliary Operations

In the hierarchical forest, any ellipsoid in an object can be used to represent the object because all the ellipsoids in the object can be accessed via the circular linked-list. Furthermore, the representative (i.e., the root) of the object can be accessed by following the parent pointers. With this note in mind, we introduce the basic operations.

Operation `overlap`: Given objects $\mathbb{O}_1$ and $\mathbb{O}_2$, this operation determines if $\mathbb{O}_1$ and $\mathbb{O}_2$ overlap. Alg. 3.7 shows the algorithmic implementation of `overlap` with the delimiting box test. Note that Alg. 3.7 l.4 utilizes the circular linked-lists of the enhanced trees to iterate over each ellipsoid in constant time.

Operation $\texttt{merge}_\texttt{H}$: Given objects $\mathbb{O}_1$ and $\mathbb{O}_2$, this operation unifies the enhanced trees corresponding to $\mathbb{O}_1$ and $\mathbb{O}_2$ into a single enhanced tree in the hierarchical forest. First of all, the roots/representatives of the two objects are retrieved using the `findRoot` operation. Second, the aforementioned roots are linked via a call to the $\texttt{link}_\texttt{H}$ operation. Third, to make the ellipsoids in the new enhanced tree form a circular linked-list, the circular linked-lists associated with $\mathbb{O}_1$ and $\mathbb{O}_2$ are unified by *swapping* the next pointers of the roots. Finally,

the delimiting box of the new object is adjusted so that it encompasses the delimiting boxes of $\mathbb{O}_1$ and $\mathbb{O}_2$.The operation is conducted in-place, avoiding unnecessary data copying or moving

Operation $\texttt{findRoot}_\texttt{H}$: Given an ellipsoid $\texttt{e}$, $\texttt{findRoot}_\texttt{H}$ traverses the chain of parent pointers until it reaches the root of the object in which $\texttt{e}$ is a member. A recursive implementation of the $\texttt{findRoot}_\texttt{H}$ operation with the path-compression heuristic is provided in Alg. 3.7, where, as the recursion unwinds on a path to a root, the parent pointers start pointing to the root. Furthermore, given an ellipsoid in an object $\mathbb{O}$, $\texttt{findRoot}_\texttt{H}$ returns the root ellipsoid in $\mathbb{O}$.

Operation $\texttt{link}_\texttt{H}$: This operation links two ellipsoids $\texttt{e}$ and $\texttt{e}'$, as the roots of two distinct objects, using the union-by-rank heuristic. To that end, $\texttt{link}_\texttt{H}$ sets the parent pointer of the one with the lower rank to the other one (i.e., attaching the shorter tree to the taller tree). If $\texttt{e}$ and $\texttt{e}'$ have the same rank, then one of them is chosen to be the new root, and its rank gets incremented. Alg. 3.7 outlines the $\texttt{link}_\texttt{H}$ operation.

3.6.3 Flat Ellipsoid Forest

As we explained in Definition 3.2, the flat forest extends the disjoint set data structure by the `ellipsoidLinking` operation. The flat forest allows concurrent wait-free execution of `ellipsoidLinking` operations in any order by utilizing fine-grained synchronization primitives as proposed in [109].

Compound Operation

Operation `ellipsoidLinking`: Given split-summaries φ_i and φ_j, this operation checks whether each ellipsoid pair $\langle \texttt{e}, \texttt{e}' \rangle$, where $\texttt{e}$ belongs to φ_i and $\texttt{e}'$ belongs to φ_j, overlaps. In that case, it merges the objects associated with $\texttt{e}$ and $\texttt{e}'$. Alg. 3.8 outlines the algorithmic implementation of `ellipsoidLinking`.

Algorithm 3.8 Operation `ellipsoidLinking` in a flat ellipsoid forest

```
1: procedure ellipsoidLinking(φi,φj)
2:   for e ∈ φi.list & e′ ∈ φj.list do
3:     if e and e′ overlap then
4:       merge_F(e, e′)
```

Auxiliary Operations

Operation $\texttt{findRoot}_\texttt{F}$: Given an ellipsoid $\texttt{e}$, this operation follows the parent pointers from $\texttt{e}$ until it reaches the root of the object in which $\texttt{e}$ belongs.

Operation $\texttt{merge}_\texttt{F}$: Given two ellipsoids $\texttt{e}$ and $\texttt{e}'$, this operation merges the trees in the forest that are associated with $\texttt{e}$ and $\texttt{e}'$. First of all, utilizing the $\texttt{findRoot}_\texttt{F}$ operation, the representatives of the objects associated with $\texttt{e}$ and $\texttt{e}'$ are found. Afterwards, the parent pointer of the object representative with the lower `ID` value gets linked to the object representative with the higher `ID` value. As there are concurrent accesses to the elements in the forest, there might be other threads that link the aforementioned parent pointer to another ellipsoid. Therefore, the implementation shown in Alg. 3.9 utilizes `CAS` to atomically

update the parent pointers. If the `CAS` operation fails when changing a parent pointer, it means that the parent pointer was already changed by some other thread (executing an overlapping `ellipsoidLinking` operation involving some ellipsoid(s) belonging to the same object(s)) in the meantime. Therefore, the roots of the associated objects are recalculated, and then the same mechanism tries to link them. The aforementioned steps continue in the retry loop (shown in Alg. 3.9 l.6-l.16) until the roots of the associated objects get linked (by any of the threads). The operation is conducted in-place, avoiding unnecessary data copying or moving.

Algorithm 3.9 Auxiliary operations in a flat ellipsoid forest. The last executed step marked by an asterisk gives the linearization point. Adapted from [109].

```
1:  procedure findRoot_F(e)
2:      while e.parent ≠ ∅* do
3:          e:=e.parent
4:      return e
5:  procedure merge_F(e, e′)
6:      while true do
7:          e := findRoot_F(e)
8:          e′ := findRoot_F(e′)
9:          if e.ID < e′.ID then
10:             if CAS(e.parent, ∅, e′)* then
11:                 return
12:         else if (e.ID == e′.ID)* then
13:             return
14:         else if e.ID > e′.ID then
15:             if CAS(e′.parent, ∅, e)* then
16:                 return
```

3.6.4 Discussion on System Aspects

The proposed algorithmic descriptions of PARMA-CC algorithms incorporate a simple *scheduler* [110] for parallel execution of tasks. Such a choice allows us to uncover the algorithmic properties of the design space and as well as the behaviour of the ellipsoid forests. In general, parallel execution of tasks in PARMA-CC algorithms can be scheduled using any off-the-shelf parallelization library, such as OpenMP [111], TBB [112], and Cilk [113]. Using such parallelization libraries is orthogonal to the scope of this work as it introduces new aspects and trade-offs to the study. Nevertheless, using such libraries can facilitate scheduling and executing finer parallel tasks. For instance, each invocation of `ellipsoidLinking` operation can be decomposed into several parallelization tasks. As shown in Alg. 3.8, there is no dependency between the iterations of the for loop in `ellipsoidLinking`; therefore, each iteration of the aforementioned for loop can be performed in parallel.

3.7 Analysis

We provide an analytical study of the PARMA-CC algorithms. Notation: Let γ be an upper bound on the number of locally detected clusters in a split of data. For an object $\mathbb{O}$, let $|\mathbb{O}|$, i.e., size of $\mathbb{O}$, be the total number of ellipsoids in $\mathbb{O}$. Considering a hierarchical PARMA-CC algorithm, for a map $\mathbb{M}$, let $\|\mathbb{M}\|$ the number of all the ellipsoids in $\mathbb{M}$. Table 3.2 summarizes the notations in this section.

Table 3.2: Table of Notation

$\mathtt{N}$	$\triangleq$	number of points in the input dataset	$\lvert\mathbb{O}\rvert$	$\triangleq$	number of ellipsoids in object $\mathbb{O}$
$\mathtt{K}$	$\triangleq$	number of threads	$\lVert\mathbb{M}\rVert$	$\triangleq$	sum of number of ellipsoids in objects in $\mathbb{M}$
$\mathtt{S}$	$\triangleq$	number of data splits			
γ	$\triangleq$	number of local clusters in a data split	$\alpha(.)$	$\triangleq$	inverse Ackermann function

3.7.1 Ellipsoid Forest Analysis

Hierarchical Ellipsoid Forest

Lemma 3.1. *[Adapted from Lemma 21.13 in [108]] The worst-case and amortized time complexity of each* $\mathtt{findRoot}_H$ *operation is respectively* $\mathcal{O}(\log(\gamma\mathtt{S}))$ *and* $\mathcal{O}(\alpha(\gamma\mathtt{S}))$*, where* $\alpha(.)$ *is the inverse Ackermann function.*

Note that $\alpha(.)$ is a very slowly growing function where $\alpha(x) < 5$ for $x < 10^{80}$.

Lemma 3.2. *The worst-case time complexity of each* `overlap` *operation on objects* $\mathbb{O}_1$ *and* $\mathbb{O}_2$ *is* $\mathcal{O}(|\mathbb{O}_1||\mathbb{O}_2|)$*.*

Lemma 3.3. *The worst-case and amortized time complexity of each* $\mathtt{merge}_H$ *operation is respectively* $\mathcal{O}(\log(\gamma\mathtt{S}))$ *and* $\mathcal{O}(\alpha(\gamma\mathtt{S}))$*.*

Proof. A $\mathtt{merge}_\mathtt{H}$ calls (i) three $\mathtt{findRoot}_\mathtt{H}$ operations, (ii) one $\mathtt{link}_\mathtt{H}$ operation, and (iii) swapping the values of two pointers. According to Lemma 3.1, the worst-case and amortized time complexity of (i) is respectively $\mathcal{O}(\log(\gamma\mathtt{S}))$ and $\mathcal{O}(\alpha(\gamma\mathtt{S}))$. (ii) and (iii) are performed with $\mathcal{O}(1)$ time complexity. □

Lemma 3.4. *The worst-case time and amortized time complexities of each* `mapCombining` *on* $\mathbb{M}_i$ *and* $\mathbb{M}_j$ *are bounded from above by* $\mathcal{O}\left(\log(\gamma\mathtt{S}).||\mathbb{M}_i||.||\mathbb{M}_j||\right)$ *and* $\mathcal{O}\left(\alpha(\gamma\mathtt{S}).||\mathbb{M}_i||.||\mathbb{M}_j||\right)$*, respectively.*

The above follows from deriving an upper bound on the summation of time complexities of operations `overlap` and $\mathtt{merge}_\mathtt{H}$ as performed by the `mapCombining` operation. Note that the bound for the amortized complexity is loose for two reasons: (i) As soon as $\mathbb{O}$ and $\mathbb{O}'$ are found to have overlapping ellipsoids, `overlap` returns true without further investigation of the remaining cases, see Alg. 3.7 l.5-6. (ii) When objects $\mathbb{O}$ and $\mathbb{O}'$ do not overlap, with high probability, the delimiting boxes of $\mathbb{O}$ and $\mathbb{O}'$ do not overlap either; therefore, saving the comparisons of ellipsoids in $\mathbb{O}$ and $\mathbb{O}'$.

Flat Ellipsoid Forest

Lemma 3.5. *[adapted from Theorem 1 in [109]] Any concurrent execution order of* $\mathtt{findRoot}_F$ *and* $\mathtt{merge}_F$ *is linearizable and wait-free.*

Lemma 3.6. *[adapted from Theorem 2 in [109]] The probability that each* $\mathtt{findRoot}_F$ *and each* $\mathtt{merge}_F$ *perform* $\mathcal{O}\left(\log\left(\gamma\mathtt{S}\right)\right)$ *steps is at least* $1 - \frac{1}{\gamma\mathtt{S}}$*.*

Lemma 3.7. *The expected asymptotic time complexity of each* $\mathtt{findRoot}_F$ *and each* $\mathtt{merge}_F$ *is* $\mathcal{O}\left(\log\left(\gamma\mathtt{S}\right)\right)$*.*

Proof. Based on Lemma 3.6, the probability that each $\texttt{findRoot}_\texttt{F}$ and each $\texttt{merge}_\texttt{F}$ perform $\Theta(\gamma\texttt{S})$ steps (the maximum possible height of a tree in the ellipsoid forest) is at most $\frac{1}{\gamma\texttt{S}}$. Therefore, the expected time complexity of each $\texttt{findRoot}_\texttt{F}$ and $\texttt{merge}_\texttt{F}$ is less than or equal to $(1-\frac{1}{\gamma\texttt{S}})\times\mathcal{O}\left(\log\left(\gamma\texttt{S}\right)\right)+\frac{1}{\gamma\texttt{S}}\times\times(\gamma\texttt{S})$, yielding bound $\mathcal{O}\left(\log\left(\gamma\texttt{S}\right)\right)$. □

Lemma 3.8. *The expected asymptotic time complexity of each* `ellipsoidLinking` *operation is* $\mathcal{O}\left(\gamma^2\log\left(\gamma\texttt{S}\right)\right)$.

Proof. Consider `ellipsoidLinking` on two given maps. Due to the linearity of expectation, the expected time complexity of `ellipsoidLinking` is the sum of expected time complexities of the $\texttt{merge}_\texttt{F}$ operations that it performs. Consider two given maps. The maximum number of times that `ellipsoidLinking` can perform the $\texttt{merge}_\texttt{F}$ on the two maps is at most $\mathcal{O}(\gamma^2)$ times (the number of pairs of ellipsoids in the two maps), where each $\texttt{merge}_\texttt{F}$ has expected time complexity of $\mathcal{O}\left(\log\left(\gamma\texttt{S}\right)\right)$ according to Lemma 3.7. □

3.7.2 Safety and Completeness Properties

Lemma 3.9. *Operations and* `mapCombining` *and* `ellipsoidLinking` *satisfy the commutative and associative properties.*

The above follows from the descriptions and the algorithmic implementations introduced in § 3.6.2 and § 3.6.3.

Lemma 3.10. *For any concurrent execution of a PARMA-CC algorithm, there exists a sequential execution that produces an equivalent result.*

Proof. We argue how to build an equivalent sequential execution corresponding to a concurrent execution of a PARMA-CC algorithm. Similar to the concurrent execution, the equivalent sequential algorithm splits the input dataset into `S` splits and operates in three matching phases, except for the synchronization details, which are not needed in the equivalent sequential execution. Regarding phase I, the signaling mechanism in $\texttt{PARMA}_\texttt{H}$ (shown in Alg. 3.2 l.9), updating the status values in $\texttt{PARMA}_\texttt{F}$ (shown in Alg. 3.3 l.7-8), insertions in `Q` in $\texttt{FLEXI-PARMA}_\texttt{H}$ (shown in Alg. 3.4 l.11), and updating the status values in $\texttt{FLEXI-PARMA}_\texttt{F}$ (shown in Alg. 3.5 l.9-10) are not needed in the equivalent sequential algorithm. Note that besides the aforementioned synchronization details, the rest of the operations in phase I of a PARMA-CC execution are performed in a data parallel fashion. Therefore, in phase I, the equivalent sequential algorithm can perform the local clustering tasks and create the split-summaries in any arbitrary order. The same argument also holds regarding phase III of the equivalent sequential algorithm. We explain how to construct the rest of the equivalent sequential execution (i.e., phase II) for the hierarchical and flat PARMA-CC algorithms in the following:

Hierarchical: A hierarchical PARMA-CC algorithm performs `mapCombining` (see Alg. 3.6) operations in the hierarchical forest according to hierarchy $\mathbb{H}$, where $\mathbb{H}$ can either be predetermined as in $\texttt{PARMA}_\texttt{H}$ or be dynamically determined as in $\texttt{FLEXI-PARMA}_\texttt{H}$. In either case, `mapCombining` operations are performed with respect to the following rules: (P1) `mapCombining` operations corresponding to disjoint subtrees in $\mathbb{H}$ can be performed in parallel. (P2) `mapCombining`

operations which have an ancestor-descendant relation in $\mathbb{H}$ never modify the same sets in the forest simultaneously. (P3) Each ellipsoid belongs to only one object both before and after a `mapCombining` operation. (P4) All pairs of objects that have overlapping ellipsoids are merged in the final map. Therefore, in phase II, the equivalent sequential algorithm can sequentially perform `mapCombining` operations following any arbitrary hierarchy $\mathbb{H}$ and get the same set of objects because operation `mapCombining` satisfies the commutative and associative properties (see Lemma 3.9).

Flat: The threads in a flat PARMA-CC algorithm perform `ellipsoidLinking` (see Alg. 3.8) operations on all pairs of split-summaries (i.e., elements of `V` as defined in Definition 3.3). We show in the following that any arbitrary (due to concurrency) inter-leaving of `ellipsoidLinking` operations, results in the same set of objects.

[A] Each `ellipsoidLinking` operation in `V` is performed exactly once. The latter holds because each thread tries to atomically book available an `ellipsoidLinking` operation via performing `CAS` on the corresponding status value (see Definition 3.3), as long as there are available `ellipsoidLinking` operations left.

[B] As the threads perform `ellipsoidLinking` operations, concurrent executions of operation $\texttt{merge}_{\texttt{F}}$ might be performed.

[C] As any concurrent execution of $\texttt{merge}_{\texttt{F}}$ is linearizable (see Lemma 3.5), and operation `ellipsoidLinking` satisfies the commutative and associative properties (see Lemma 3.9), the same set of objects get formed in the ellipsoid forest regardless of linearization of the $\texttt{merge}_{\texttt{F}}$ operations.

Based on the sequence of arguments in (1), (2), and (3), any concurrent execution of `ellipsoidLinking` operations on all pairs of split-summaries results in the same set of objects. Therefore, in phase II, the equivalent sequential algorithm can sequentially perform `ellipsoidLinking` in any arbitrary order and get the same set of objects. □

Definition 3.5 (The Completeness Property)**.** *An ellipsoid forest satisfies the completeness property when the following condition holds for each pair of ellipsoids* $\langle \texttt{e}_\texttt{i}, \texttt{e}_\texttt{j} \rangle$ *in the forest: The pair* $\langle \texttt{e}_\texttt{i}, \texttt{e}_\texttt{j} \rangle$ *is directly or indirectly overlapping (see Definition 3.1) if and only if there exists an object* $\mathbb{O}$ *such that* $\texttt{e}_\texttt{i} \in \mathbb{O}$ *and* $\texttt{e}_\texttt{j} \in \mathbb{O}$*.*

Lemma 3.11 (Completeness in $\texttt{PARMA}_\texttt{H}$)**.** *By the end of phase II in* $\texttt{PARMA}_\texttt{H}$*, the completeness property holds in the associated hierarchical ellipsoid forest.*

Proof. We first prove the statement in the following direction: If the pair $\langle \texttt{e}_\texttt{i}, \texttt{e}_\texttt{j} \rangle$ is directly or indirectly overlapping, then, by the end of phase II, there exists an object $\mathbb{O}$ such that $\texttt{e}_\texttt{i} \in \mathbb{O}$ and $\texttt{e}_\texttt{j} \in \mathbb{O}$. To that end, consider phase I, when $\texttt{e}_\texttt{i}$ is a member of $\mathbb{O}_{i'}$ in map $\mathbb{M}_i$, and $\texttt{e}_\texttt{j}$ is a member of $\mathbb{O}_{j'}$ in map $\mathbb{M}_j$. If the pair $\langle \texttt{e}_\texttt{i}, \texttt{e}_\texttt{j} \rangle$ is directly overlapping, then $\mathbb{O}_{i'}$ and $\mathbb{O}_{j'}$ are merged when `mapCombining` operation is performed on the maps containing $\texttt{e}_\texttt{i}$ and $\texttt{e}_\texttt{j}$ (such a `mapCombining` operation is guaranteed to exist as $\mathbb{H}$ is a spanning tree, see § 3.3.3). On the other hand, suppose the pair $\langle \texttt{e}_\texttt{i}, \texttt{e}_\texttt{j} \rangle$ is indirectly overlapping via ellipsoid $\texttt{e}_\texttt{k}$ (i.e., ellipsoids in each of the pairs $\langle \texttt{e}_\texttt{i}, \texttt{e}_\texttt{k} \rangle$ and

$\langle \mathtt{e_j}, \mathtt{e_k} \rangle$ are directly overlapping), where $\mathtt{e_k}$ belongs to object $\mathbb{O}_{k'}$, at the end of phase I, in $\mathbb{M}_k$. After `mapCombining` operations are performed on $\mathbb{M}_i$, $\mathbb{M}_j$, and $\mathbb{M}_k$ in the order specified by $\mathbb{H}$, there will be an object containing $\mathtt{e_i}$, $\mathtt{e_j}$, and $\mathtt{e_k}$. The latter holds regardless of the hierarchy specified by $\mathbb{H}$ because the `mapCombining` operation satisfies the commutative property (see Lemma 3.9). This argument can be extended inductively to cover all the cases in which $\mathtt{e_i}$ and $\mathtt{e_j}$ are indirectly overlapping.

Now we prove the statement in the opposite direction: If there exists an object $\mathbb{O}$ such that $\mathtt{e_i} \in \mathbb{O}$ and $\mathtt{e_j} \in \mathbb{O}$, then the pair $\langle \mathtt{e_i}, \mathtt{e_j} \rangle$ is directly or indirectly overlapping. Towards a contradiction, suppose the pair $\langle \mathtt{e_i}, \mathtt{e_j} \rangle$ is neither directly nor indirectly overlapping, but $\mathtt{e_i} \in \mathbb{O}$ and $\mathtt{e_j} \in \mathbb{O}$. The latter implies that, at some point in phase II, the `mapCombining` operation combined non-overlapping objects, a contradiction.

□

Lemma 3.12 (Completeness in $\mathtt{PARMA_F}$)**.** *By the end of phase II in* $\mathtt{PARMA_F}$*, the completeness property holds in the associated flat ellipsoid forest.*

Proof. We first prove the statement in the following direction: If the pair $\langle \mathtt{e_i}, \mathtt{e_j} \rangle$ is directly or indirectly overlapping, then there exists an object $\mathbb{O}$ such that $\mathtt{e_i} \in \mathbb{O}$ and $\mathtt{e_j} \in \mathbb{O}$. To that end suppose $\mathtt{e_i} \in \varphi_\mathtt{i}$ and $\mathtt{e_j} \in \varphi_\mathtt{j}$. If the pair $\langle \mathtt{e_i}, \mathtt{e_j} \rangle$ is directly overlapping, then, through the call to `ellipsoidLinking`($\varphi_\mathtt{i}$, $\varphi_\mathtt{j}$), $\mathtt{e_i}$ and $\mathtt{e_j}$ will become members of the same object. On the other hand, if the pair $\langle \mathtt{e_i}, \mathtt{e_j} \rangle$ is indirectly overlapping with just an ellipsoid $\mathtt{e_k} \in \varphi_\mathtt{k}$ in between (i.e., ellipsoids in each of the pairs $\langle \mathtt{e_i}, \mathtt{e_k} \rangle$ and $\langle \mathtt{e_j}, \mathtt{e_k} \rangle$ are directly overlapping), then there will be an object containing $\mathtt{e_i}$ and $\mathtt{e_j}$ (as well as $\mathtt{e_k}$) after `ellipsoidLinking`($\varphi_\mathtt{i}$, $\varphi_\mathtt{k}$) and `ellipsoidLinking`($\varphi_\mathtt{j}$, $\varphi_\mathtt{k}$) are completed. This argument can be inductively extended to cover all the cases in which $\mathtt{e_i}$ and $\mathtt{e_j}$ are indirectly overlapping.

The proof in the opposite direction is made with contradiction, similar to the one provided in the proof of Lemma 3.11. □

Lemma 3.13 (Completeness in Flexi PARMA-CC Algorithms)**.** *At the end of phase II, the ellipsoid forest in a flexi PARMA-CC satisfies the completeness property.*

Proof. For a given hierarchical/flat flexi PARMA-CC algorithm operating on `S` splits, consider a hierarchical/flat basic PARMA-CC algorithm that operates with `K`=`S` threads. The two algorithms produce equivalent ellipsoid forests because both `mapCombining` and `ellipsoidLinking` satisfy the commutative property. Therefore, the ellipsoid forest at the end of phase II of the flexi PARMA-CC algorithm satisfies the completeness property similar to the basic PARMA-CC algorithm (based on Lemma 3.11 and Lemma 3.12). □

Corollary 3.1. *With fixed* `minPts`*,* ϵ*, and* `S`*, PARMA-CC algorithms yield the same clustering for the same input dataset.*

3.7.3 Completion Time of PARMA-CC Algorithms

Here we analyze the completion time behaviour of the algorithms in the PARMA-CC family.

Assumptions:

- As D can contain several hundreds of thousands of points, but the number of splits is limited to a few hundreds, we assume $\mathtt{N} \gg \mathtt{S} \geq \mathtt{K}$. Furthermore, we assume $\mathtt{N} \gg \gamma$ because a local cluster can typically contain a large number of points.
- The local clustering algorithm (for instance PCL-EC or DBSCAN) uses a kd-tree to perform ϵ-neighbourhood queries.
- The total number of ellipsoids in a an ellipsoid forest ($\gamma\mathtt{S}$) is smaller than 10^{80} for all the possible use-cases. Therefore, all occurrences of the inverse Ackermann function are substituted with $\mathcal{O}(1)$.

Lemma 3.14. *The following statements hold regarding the completion time of different phases of a PARMA-CC algorithm:*

- *The expected completion time of phase I is* $\mathcal{O}(\frac{\mathtt{N}}{\mathtt{K}} \log(\frac{\mathtt{N}}{\mathtt{S}}))$.
- *The expected completion time of phase II of a hierarchical PARMA-CC algorithm is* $\mathcal{O}(\gamma^2\mathtt{S}^2)$.
- *The expected completion time of phase II of a flat PARMA-CC algorithm is* $\mathcal{O}\left(\frac{\gamma^2\mathtt{S}^2}{\mathtt{K}} \log(\gamma\mathtt{S})\right)$.
- *The expected completion time of phase III in a hierarchical PARMA-CC algorithm is* $\mathcal{O}(\frac{\mathtt{N}}{\mathtt{K}})$.
- *The expected completion time of phase III in a flat PARMA-CC algorithm is* $\mathcal{O}\left(\frac{\mathtt{N}}{\mathtt{K}} \log(\gamma\mathtt{S})\right)$.

Proof. We prove each statement in the following:

- We prove the statement for basic and flat PARMA-CC algorithms:
  - In case of a basic PARMA-CC algorithm ($\mathtt{S} = \mathtt{K}$): As the workload is distributed evenly among the K threads, the expected completion time of a data split clustering is $\mathcal{O}(\frac{\mathtt{N}}{\mathtt{K}} \log(\frac{\mathtt{N}}{\mathtt{K}}))$.
  - In case of a flexi PARMA-CC algorithm (S ¿ K): There are S local clustering tasks to be shared by K threads, and as each split contains N/S points, the expected completion time of a data split clustering is $\mathcal{O}(\frac{\mathtt{N}}{\mathtt{S}} \log(\frac{\mathtt{N}}{\mathtt{S}}))$. Therefore, the expected completion time of K threads concurrently performing local clustering is $\mathcal{O}\left(\frac{\mathtt{S}}{\mathtt{K}}\frac{\mathtt{N}}{\mathtt{S}} \log(\frac{\mathtt{N}}{\mathtt{S}})\right) = \mathcal{O}(\frac{\mathtt{N}}{\mathtt{K}} \log(\frac{\mathtt{N}}{\mathtt{S}}))$.

  Other computational steps in phase I of a PARMA-CC (i.e., fitting bounding ellipsoids, applying synchronization primitives, pushing elements into a queue) are asymptotically dominated by $\mathcal{O}(\frac{\mathtt{N}}{\mathtt{K}} \log(\frac{\mathtt{N}}{\mathtt{S}}))$.
- Summing up the amortized time complexities of all `mapCombining` operations (see Lemma 3.4), the expected total time complexity of all `mapCombining` operations is bounded from above by $\mathcal{O}(\gamma^2\mathtt{S}^2)$, which is a loose bound based on the proof of Lemma 3.4.

- In phase II of a flat PARMA-CC algorithm, $\mathcal{O}(\mathtt{S}^2)$ `ellipsoidLinking` tasks are performed. The aforementioned tasks are shared by `K` threads running in parallel. Considering the fine granularity of the tasks, each thread performs $\mathcal{O}(\mathtt{S}^2/\mathtt{K})$ `ellipsoidLinking` operations. The result follows from applying the linearity of expectation over the expected time complexity of each `ellipsoidLinking` operation (given in Lemma 3.8).
- We prove the statement for basic and flat PARMA-CC algorithms:
  - In case of a basic PARMA-CC algorithm (`S` = `K`): As the workload is distributed evenly among the `K` threads, each thread relabels $\mathcal{O}(\frac{\mathtt{N}}{\mathtt{K}})$ points.
  - In case of a flexi PARMA-CC algorithm (`S` ¿ `K`): There are `S` local relabeling tasks to be shared by `K` threads, and as each split contains `N`/`S` points, each thread relabels $\mathcal{O}\left(\frac{\mathtt{S}}{\mathtt{K}}\frac{\mathtt{N}}{\mathtt{S}}\right) = \mathcal{O}(\frac{\mathtt{N}}{\mathtt{K}})$ points.

  The amortized time complexity of relabeling each point is $\mathcal{O}(1)$ because finding the root of the associated tree, via performing $\mathtt{findRoot}_\mathtt{H}$ (Lemma 3.1), and then retrieving the root's ID are the only required steps for each point. The expected completion time can be driven by taking a summation over the amortized time complexities of relabeling each point.
- This statement is proven similar to the previous one. Due to the linearity of expectation, summing the expected completion times of $\mathcal{O}(\frac{\mathtt{N}}{\mathtt{K}})$ $\mathtt{findRoot}_\mathtt{F}$ operations (given in Lemma 3.7) yields the result.

□

Observation 3.2. *Lemma 3.14 indicates the following trade-off between the expected completion time of phase I and the expected completion time of phase II: the dominating factor in the completion time of a PARMA-CC algorithm, i.e., the local clustering in phase I, can be reduced by increasing* `K` *and/or* `S`*. However, too large values for* `K` *and/or* `S` *increase the expected completion time of phase II.*

Theorem 3.1. *The expected completion time of a PARMA-CC algorithm under the given assumptions is* $\mathcal{O}(\frac{\mathtt{N}}{\mathtt{K}}\log(\frac{\mathtt{N}}{\mathtt{S}}))$.

Proof. The theorem follows from taking the dominating asymptotic term in Lemma 3.14. □

3.7.4 On Shared Memory Accesses and Contention

As discussed in § 3.6.1, the operations on the data structure are in-place, avoiding unnecessary copies and moves of data. Regarding contention on the shared memory in different PARMA-CC algorithms, note that there is none in $\mathtt{PARMA}_\mathtt{H}$ because the computations that each thread performs follow a predetermined partial order that ensures that concurrent operations touch disjoint data only. The number of occasions in which shared memory contention can take place in $\mathtt{FLEXI\text{-}PARMA}_\mathtt{H}$ is proportionate to `S`, i.e., the number of shared tasks. On the other hand, the number of shared tasks in flat PARMA-CC algorithms is proportionate to $\mathtt{S}^2$, determined by the number of `ellipsoidLinking` operations.

Note that this shared memory contention discussion is complementary to the expected completion time analysis in Lemma 3.14, which, among other factors, takes into account the expected number of retries that have to be performed because of memory contention, where necessary.

Note when `X` threads concurrently perform a `CAS` operation on a memory location, only one of them succeeds and $\texttt{X} - 1$ threads fail. Therefore, to measure memory contention, we consider the average ratio of failed CAS operations to the total number of invoked CAS operations. Exact measurements for the ratio of failed CAS operations to the total number of invoked CAS operation is data-dependent and execution-dependent. In the worst-case, the aforementioned ratio can be as large as $1 - \frac{1}{\texttt{K}}$, indicating one successful `CAS` against $\texttt{K} - 1$ unsuccessful `CAS` for all the invocations. In the algorithmic implementation of PARMA-CC, some invocations of `CAS` operations can be avoided; e.g., a thread does not need to invoke a `CAS` to book a local clustering task which is already booked or completed (see Alg. 3.4 l.7 and Alg. 3.5 l.6). The same also applies for booking local clustering tasks and `ellipsoidLinking` operations. We empirically measure the aforementioned ratio in § 3.9.5.

3.8 Discussion on the Utilization and Building Components

3.8.1 On which PARMA-CC Algorithm to Choose

Let *inter-split overlap* refer to the amount of overlap between local clusters in different splits. With high inter-split overlap, utilizing the hierarchical forest can give higher scalability compared to utilizing the flat forest. First of all, as the inter-split overlap increases, the average number of ellipsoids in different objects increases. Consecutively, the computational savings of the delimiting-box test increase as a result of skipping the comparison of ellipsoids in non-overlapping objects. On the other hand, the amount of concurrent updates on overlapping elements in a flat forest is directly proportional to the inter-split overlap. Therefore, threads performing `ellipsoidLinking` in a flat forest have to retry (see Alg. 3.9 l.6) for more number of times for successful linking as the inter-split overlap increases.

Observation 3.3. *With the splitting mechanism outlined in § 3.3.2 and input data having a* spatio-temporal *locality (e.g., an angularly sorted LIDAR point cloud), the inter-split overlap is low. Therefore, we expect the flat PARMA-CC algorithms to scale better under the aforementioned conditions. On the contrary, we expect the hierarchical PARMA-CC algorithms to scale better on arbitrarily ordered datasets that exhibit high inter-split overlap. Table 3.3 summarises the PARMA-CC algorithms.*

3.8.2 Use Cases Implying Extensions

PARMA-CC's summaries can be used to efficiently answer a range of queries. We demonstrate the latter by studying two common queries, for which we study and compare how PARMA-CC and a classical approach can be used.

Table 3.3: Algorithms in the PARMA-CC family (SP stands for synchronization primitives)

Algorithm	Ellipsoid Forest/ Combine Order	Basic/ Flexi	Synchronization			Preferred Data Properties
			Phase I	Phase II	Phase III	
PARMA$_H$	Hierarchical/ predetermined	Basic	-	SP	-	arbitrarily ordered
PARMA$_F$	Flat/ dynamic	Basic	-	SP	-	spatio-temporal (e.g. LIDAR)
FLEXI-PARMA$_F$	Flat/ dynamic	Flexi	SP	SP	SP	spatio-temporal (e.g. LIDAR)
FLEXI-PARMA$_H$	Hierarchical/ dynamic	Flexi	Queue+SP	SP	SP	arbitrarily ordered

Algorithm 3.10 Answering queries using PARMA-CC's summaries

```
1: procedure predictLabel(q)
2:     for i ∈ {1, ··· , S} do
3:         for e ∈ φ_i.list do
4:             if q falls within e then
5:                 return findRoot(e).ID
6:     return noise
7: procedure distanceToObject(𝕆, q)
8:     return min_{e∈𝕆} {distance between q and e}
```

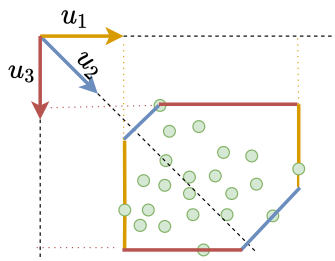


Figure 3.4: Polyhedron fitting

Predicting the clustering label of a new point q based on the existing clusters: The latter can be useful in evolving sets. A classical approach might decide about q's clustering label by considering the clustering labels of of q's nearest neighbours in D. Using a kd-tree, nearest neighbour queries have expected and worst-case time complexities of $\mathcal{O}(\log \mathtt{N})$ and $\mathcal{O}(\mathtt{N}^2)$, respectively. On the other hand, the approach leveraging the summaries of a PARMA-CC algorithm can assign q the unique ID of $\mathbb{O}_\mathtt{q}$'s root, where $\mathbb{O}_q$ is the object in which q geometrically falls. The latter is shown as operation predictLabel in Alg. 3.10 l.1-6. As the total number of ellipsoids is $\gamma\mathtt{S}$, predictLabel's worst-case time complexity is $\mathcal{O}(\gamma\mathtt{S})$. Note that, in general, $\gamma\mathtt{S}$ is much smaller than N.

Approximating the distance of a given point q to the nearest point in cluster c: With time complexity $\mathcal{O}(|\mathtt{c}|)$, a classical approach calculates the distance of each point in c to q and returns the smallest one. On the other hand, the approach levering PARMA-CC's summaries computes the distance of q to each ellipsoid in $\mathbb{O}_\mathtt{c}$, where $\mathbb{O}_\mathtt{c}$ is a PARMA-CC object corresponding to cluster c. The latter is shown as operation distanceToObject in Alg. 3.10. The distance between a point and an ellipsoid is determined in $\mathcal{O}(1)$ using the method in [114]. Therefore, distanceToObject's time complexity is $\mathcal{O}(|\mathbb{O}|)$.

3.8.3 On Volumetric Summarization Methods

Besides the bounding ellipsoid summarization method, PARMA-CC algorithms can utilize other geometric summarization methods such as axis-aligned bounding boxes (AABBs) [115] or oriented bounding boxes (OBBs) [116]. More

generally, we consider *bounding polyhedrons*. Figure 3.4, shows an example cluster of points (represented by green circles) and a corresponding bounding polyhedron. We characterize a bounding polyhedron by a set of normal vectors $\mathtt{u_i}$ for $\mathtt{i} \in \{1, \cdots, \mathtt{F}\}$, where each $\mathtt{u_i}$ is a normal vector to two parallel faces in the bounding polyhedron.
Fitting a bounding polyhedron around a local cluster c: The minimum and maximum values of the orthogonal projections of points in c onto vector $\mathtt{u_i}$ determine respectively the *left* and *right* faces associated with normal vector $\mathtt{u_i}$. Note that the orthogonal projection of a point onto a vector is simply calculated by their dot product. The example in Figure 3.4 utilizes three normal vectors $\mathtt{u_1}$, $\mathtt{u_2}$, and $\mathtt{u_3}$. Note that the left and right faces associated with each normal vector are shown in the same color as the normal vector.
Determining the normal vectors: The normal vectors $\mathtt{u_i}$s can either be chosen randomly or in a systematic way to uniformly sample the unit sphere in the space. The number of vectors, F, determines the granularity of the volumetric approximation, i.e., increasing F increases the approximation accuracy but increases the cost of computing the bounding polyhedron. However, with fixed F, the cost of fitting a bounding polyhedron is constant per point in c.
Determining whether two bounding polyhedra geomterically overlap: Two polyhedra $\mathtt{P_1}$ and $\mathtt{P_2}$ are geometrically overlapping if and only if, for all $\mathtt{u_i}$s, the intervals containing the left and right faces in $\mathtt{P_1}$ and $\mathtt{P_2}$ overlap.

3.9 Evaluation

We here empirically evaluate PARMA-CC algorithms. The parameters of the study are the following: the number of threads (K), the number of splits (S), the size of the input data (N), the number of objects in the input data, the degree of inter-split overlap in the input data, and the local clustering algorithm.

3.9.1 Experiment Setup

We study the completion time of PARMA-CC algorithms in accordance with the expected completion time analysis given in Theorem 3.1, and we study the scalability of PARMA-CC algorithms in conjunction with Observation 3.1. We complement the completion time study of PARMA-CC algorithms by empirically observing the expectations raised in Observation 3.3 regarding the behaviour of the algorithms with different degrees of inter-split overlap. Furthermore, we study the ratio of the local clustering time to the completion time of the algorithms in accordance with the analytical results in Lemma 3.14 to gain insight on how the different phases contribute to the total completion time. Moreover, we measure the accuracy of the algorithms using *rand index*[4]. Finally, we complement the shared memory access and contention analysis in § 3.7.4 by empirically measuring the average ratio of failed CAS operations to the total number of invoked CAS operations.

We provide the evaluation results corresponding to PARMA-CC algorithms that utilize PCL-EC (see § 3.2) as the local clustering algorithm. Moreover, in order to evaluate PARMA-CC algorithms utilizing a density-based local

[4]https://github.com/bjoern-andres/partition-comparison

Table 3.4: Summary of the bench-marked datasets, showing the characteristics and the chosen clustering parameters for each dataset.

dataset	KITTI	FORD	GEOLIFE, shuffled GEOLIFE	MOPSI, shuffled MOPSI
`N`	40,000	150,000-300,000	1.4 million	1.2 million
inter-split overlap	low	low	medium, high	medium, high
(ϵ, `minPts`)	(0.7, 10)	(0.5, 100)	(0.001, 500), (0.001, 500/`S`)	(0.1, 500), (0.1, 500/`S`)
`S`	{2, 3, 4, 5, 10, 15, 20, 30, 36, 40, 50, 60, 70, 140}		{2, 3, 4, 5, 10, 15, 20, 30, 36, 40, 50, 60, 70, 100, 200, 400, 600}, {2, 3, 4, 5, 10, 15, 20, 30, 36, 40, 50, 60, 70}	
`K`	{2, 3, 4, 5, 10, 15, 20, 30, 36, 40, 50, 60, 70}			

clustering algorithm, we study scalability and accuracy of basic PARMA-CC algorithms utilizing DBSCAN as the local clustering algorithm. Accordingly, we compare the scalability of the aforementioned algorithms with the scalability of PDSDBSCAN (see § 3.2). By default, the presented results and discussions refer to PARMA-CC algorithms that utilize PCL-EC as the local clustering algorithm, unless otherwise stated.

Evaluation data: We use both LIDAR and GPS datasets. Regarding LIDAR data, we study a random subset of the point clouds in the KITTI dataset [91], collected by a Velodyne laser scanner in urban driving. We also study a random subset of the Ford Multi-AV Seasonal dataset [76] which is collected by a fleet of vehicles in a variety of conditions. Regarding GPS data, we choose a random subset of points in the Mopsi route dataset [8], which contains GPS readings (in terms of latitude and longitude) gathered by various users doing a wide range of activities (e.g., walking, cycling, skiing, taking a boat) mostly in Finland. We also study a random subset of the GeoLife GPS Trajectories dataset [117], containing densely recorded GPS readings by several users mostly in Beijing city. Furthermore, we study randomly shuffled versions of the GeoLife and Mopsi datasets, exhibiting high inter-split overlap. Table 3.4 gives an overview of each bench-marked dataset along with its inter-split overlap characterization.

Preprocessing: By imposing a simple threshold, we filter the ground (floor) points in the point clouds (otherwise scene objects are connected via the ground). Each filtered point cloud in the KITTI dataset contains about 40,000 points, and each filtered point cloud in our subset of the Ford Multi-AV dataset contains between 150,000 and 300,000 points. Regarding the GPS datasets, we filter sequential duplicate points (the points that correspond to when GPS readings were logged while the user was stationary). Our filtered subset of the GeoLife and Mopsi datasets contain more than 1.4 million and 1.2 million points, respectively. The size of the bench-marked datasets are shown in Table 3.4.

Parameters: Our purpose of clustering LIDAR datasets is to detect scene objects, and our purpose of clustering GPS datasets is to detect areas attracting a lot of users. To that end, we choose ϵ and `minPts` to attain valid ground truth by the baselines. We identified that the baseline achieves reasonable clustering of scene objects with $\epsilon = 0.7$ and `minPts` $= 10$ for the KITTI dataset, and so it does with $\epsilon = 0.5$ and `minPts` $= 100$ for the Ford Multi-AV dataset. Furthermore,

Table 3.5: Highlights of average elapsed completion time (in seconds) for different methods and datasets with a variety of parameters.

dataset		KITTI	FORD		GeoLife	Mopsi	
PCL		0.3598	1.8459		950	9829	
K=20	PARMA$_{\text{H}}$	0.0756	0.2304		21.6	91.7	
	PARMA$_{\text{F}}$	0.0762	0.2063		22.5	100.1	
	FLEXI-PARMA$_{\text{H}}$	0.0259	0.1074	S=140	1.3	3.24	S=600
	FLEXI-PARMA$_{\text{F}}$	0.0224	0.1016		1.5	3.5	
K=40	PARMA$_{\text{H}}$	0.0429	0.1363		13.7	35.2	
	PARMA$_{\text{F}}$	0.0418	0.1387		14.5	38.3	
	FLEXI-PARMA$_{\text{H}}$	0.0241	0.0800	S=140	0.94	2.06	S=600
	FLEXI-PARMA$_{\text{F}}$	0.0178	0.0803		0.97	2.16	
K=70	PARMA$_{\text{H}}$	0.0335	0.1038		4.9	19.9	
	PARMA$_{\text{F}}$	0.0601	0.1096		6.7	25.3	
	FLEXI-PARMA$_{\text{H}}$	0.0411	0.0827	S=140	0.8	1.54	S=600
	FLEXI-PARMA$_{\text{F}}$	0.0295	0.0756		0.85	1.58	

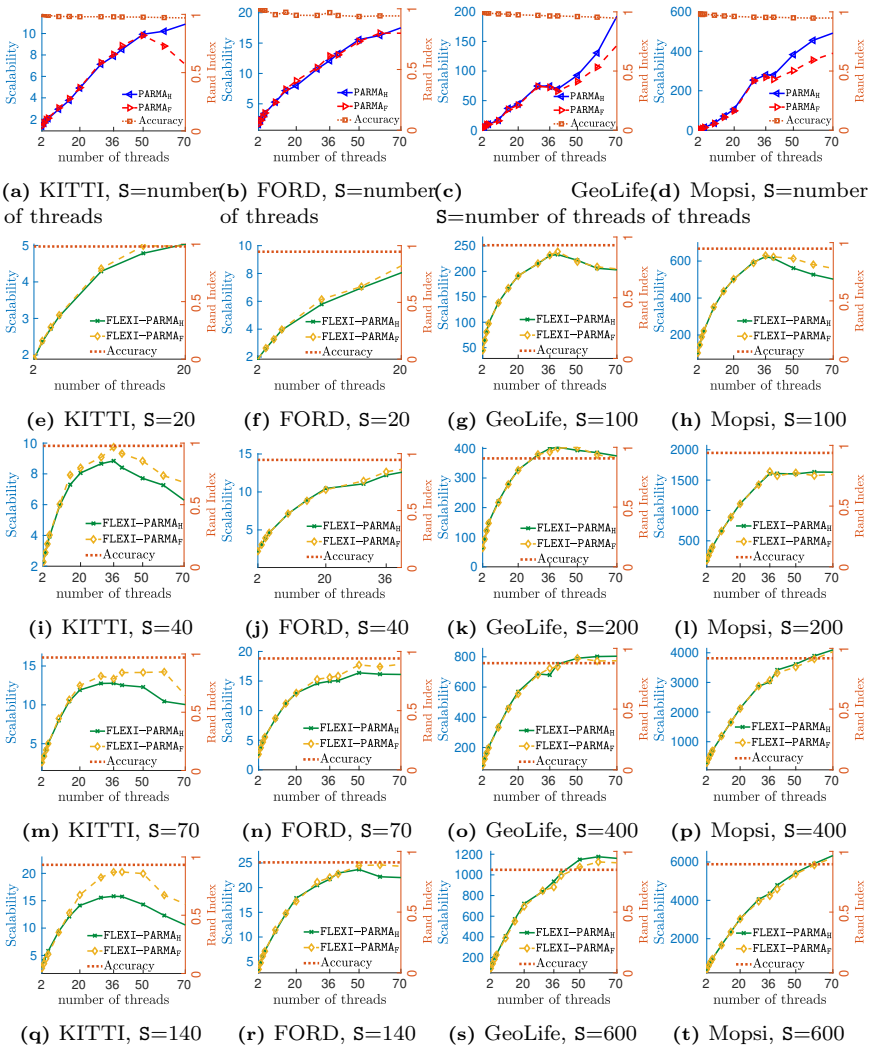
the baseline achieves reasonable clustering of GPS readings with $\epsilon = 0.1$ and `minPts` $= 500$ for the Mopsi dataset. On the other hand, as the GeoLife dataset contains much denser recordings, the baseline achieves reasonable clustering with parameters $\epsilon = 0.001$ and `minPts` $= 500$ for this dataset. For the LIDAR datasets, we execute flexi PARMA-CC algorithms choosing 20, 40, 70, and 140 for `S`. For the GPS datasets, as they contain much more points than the LIDAR datasets, we choose `S` among 100, 200, 400, and 600. We perform the experiments with up to 70 threads, except for the experiments in which `S` is less than 70, where we choose up to `S` threads. As the distribution of the points in randomly shuffled datasets becomes uniform among the splits, we adjust `minPts` with respect to `S` by using `minPts/S`. The aforementioned adjustment is a common practice, e.g., [64]. The chosen clustering parameters are summarized in Table 3.4 for each dataset.

Evaluation setup: We implemented PARMA-CC algorithms[5] in C++ and used GNU scientific library for matrix algebra. We used POSIX threads for multi-threaded programming. We used PCL's implementation of PCL-EC [6]. We employed elapsed real time to measure completion times. Experiments were run on a 2.10 GHz Intel(R) Xeon(R) E5-2695 system with 36 cores on two sockets (18 cores per socket, each core supporting two hyper-threads) and 64 GB memory in total, running Ubuntu 16.04. We only used hyper threading when there were more threads than the actual number of cores.

3.9.2 Completion Time and Scalability

Detailed scalability plots in the left Y-axes of Figure 3.5 show the scalability of PARMA-CC algorithms for different datasets with varying choices of `K` and `S` for the basic and flexi PARMA-CC algorithms. Besides, some highlights

[5] https://github.com/dcs-chalmers/PARMA-CC



(a) KITTI, S=number of threads (b) FORD, S=number of threads (c) GeoLife, S=number of threads (d) Mopsi, S=number of threads

(e) KITTI, S=20 (f) FORD, S=20 (g) GeoLife, S=100 (h) Mopsi, S=100

(i) KITTI, S=40 (j) FORD, S=40 (k) GeoLife, S=200 (l) Mopsi, S=200

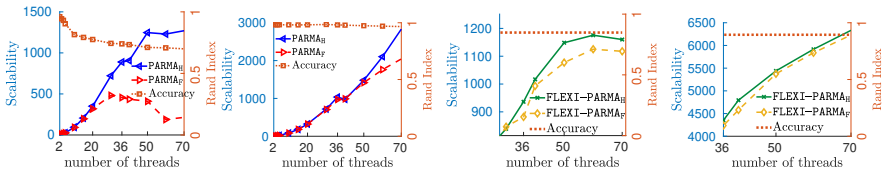
(m) KITTI, S=70 (n) FORD, S=70 (o) GeoLife, S=400 (p) Mopsi, S=400

(q) KITTI, S=140 (r) FORD, S=140 (s) GeoLife, S=600 (t) Mopsi, S=600

Figure 3.5: *Scalability and accuracy of PARMA-CC algorithms. PARMA-CC algorithms achieve the same clustering accuracy with a fixed* S.

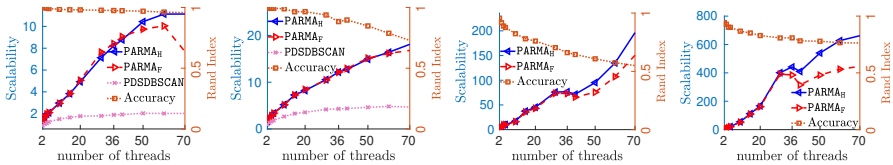
of the completion times are presented in Table 3.5 using varying number of threads and splits. Furthermore, the results of PCL-EC, as an exact sequential baseline, are included for the reference.

The results show that, with appropriate choices of S and large enough number of threads, PARMA-CC algorithms can be several orders of magnitude faster than the exact sequential baseline. Furthermore, the scalability of PARMA-CC algorithms demonstrates a super-linear behaviour with respect to K or S for the GeoLife and Mopsi datasets. As both GeoLife and Mopsi datasets have highly skewed distributions, the complexity of the exact sequential baseline is $\mathcal{O}(\mathtt{N}^2)$. The latter holds because the spatial data structure used to find ϵ-neighbourhoods is not able to operate efficiently with skewed data distributions. On the other hand, the PARMA-CC algorithms reduce the



(a) shuffled GeoLife **(b)** shuffled Mopsi **(c)** GeoLife, S=600 **(d)** Mopsi, S=600

Figure 3.6: *Scalability and accuracy of basic PARMA-CC algorithms on the shuffled GeoLife and Mopsi datasets in (a) and (b). Zoomed scalability and accuracy of flexi PARMA-CC algorithms on GeoLife and Mopsi datasets in (c) and (d).*



(a) KITTI **(b)** FORD **(c)** GeoLife **(d)** Mopsi

Figure 3.7: *Scalability and accuracy of PDSDBSCAN and basic PARMA-CC algorithms utilizing DBSCAN as the local clustering algorithm. In (c), PDSDBSCAN does not produce a proper DBSCAN clustering. In (d), PDSDBSCAN crashes as it runs out of memory. The right Y-axes show the clustering accuracy of basic PARMA-CC algorithms.*

completion time of the local clustering quadratically in K or S, by splitting the data and by approximation. With this observation in place, we discuss further the scalability behaviour of PARMA-CC algorithms in the following.

We notice that with a large enough choice of S, a flexi PARMA-CC algorithm achieves higher scalability than its basic counterpart. For example, with S being 600, the scalability of a flexi PARMA-CC algorithm is about 6 times that of a basic PARMA-CC algorithm, as shown in Figure 3.5c and Figure 3.5s. Moreover, for each dataset, we observe that the scalability of the flexi PARMA-CC algorithms tends to increase with greater S values. The latter is in accordance with Observation 3.2, stating the effect of increasing S on decreasing the completion time of local clustering. Furthermore, similar to the basic PARMA-CC algorithms, we observe the super-linear scalability of flexi PARMA-CC algorithms on the GeoLife and Mopsi datasets. With smaller sets of data we also see that, beyond some point, increasing the number of threads does not decrease further the execution time, as there is less work to be done and the benefit from distributing is opposed by the cost of coordination (Figure 3.5m, Figure 3.5q, Figure 3.5n, Figure 3.5r).

The Spatio-Temporal Properties and the Scalability of PARMA-CC Algorithms

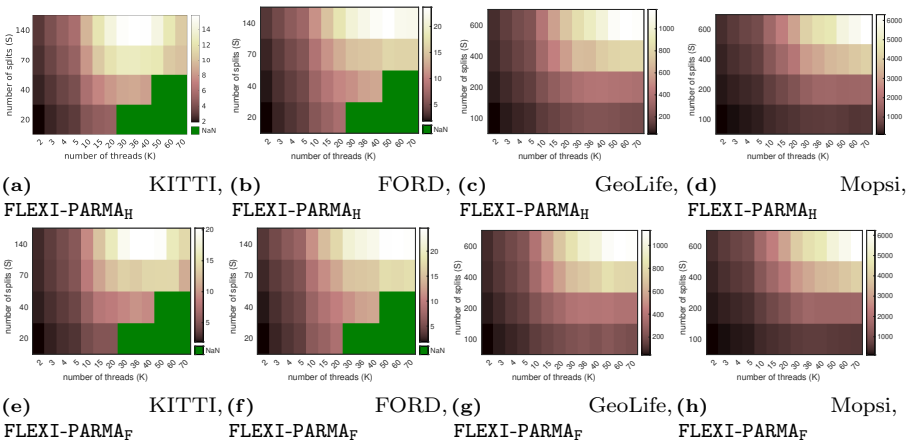
On the KITTI and FORD datasets (with low inter-split overlap), FLEXI-PARMA$_F$ has the highest scalability. The latter is shown in the left Y-axes in Fig. 3.5q-3.5r, and it is in accordance with Observation 3.3. On the other hand, FLEXI-PARMA$_H$ achieves the highest scalability on the GeoLife and Mopsi datasets, see the zoomed graphs Figure 3.6c and Figure 3.6d, respectively.

The left Y-axes in Figure 3.6a and Figure 3.6b respectively show the

scalability of the basic PARMA-CC algorithms on the randomly shuffled GeoLife and Mopsi, datasets exhibiting high inter-split overlap. The results show PARMA$_H$ typically has higher scalability than FLEXI-PARMA$_F$ on the randomly shuffled datasets. The latter is in accordance with Observation 3.3. Another important observation is that PARMA-CC algorithms typically has higher scalability on the randomly shuffled datasets. The latter holds because the splits of a randomly shuffled dataset contain approximately similar distributions, alleviating the worst-case behaviour of spatial data structures such as kd-tree.

Approximate DBSCAN Clustering and the Scalability of PARMA-CC Algorithms

The left axes in Fig 3.7a-3.7d show the average scalability of PDSDBSCAN (see § 3.2) and basic PARMA-CC algorithms that utilize DBSCAN as the local clustering algorithm on the KITTI, FORD, GeoLife, and Mopsi datasets. As the results show, even the basic PARMA-CC algorithms achieve significantly higher scalability than PDSDBSCAN. Note that PDSDBSCAN fails to produce a proper clustering in Figure 3.7c and crashes as it runs out of memory in Figure 3.7d.



(a) KITTI, **(b)** FORD, **(c)** GeoLife, **(d)** Mopsi, FLEXI-PARMA$_H$ FLEXI-PARMA$_H$ FLEXI-PARMA$_H$ FLEXI-PARMA$_H$

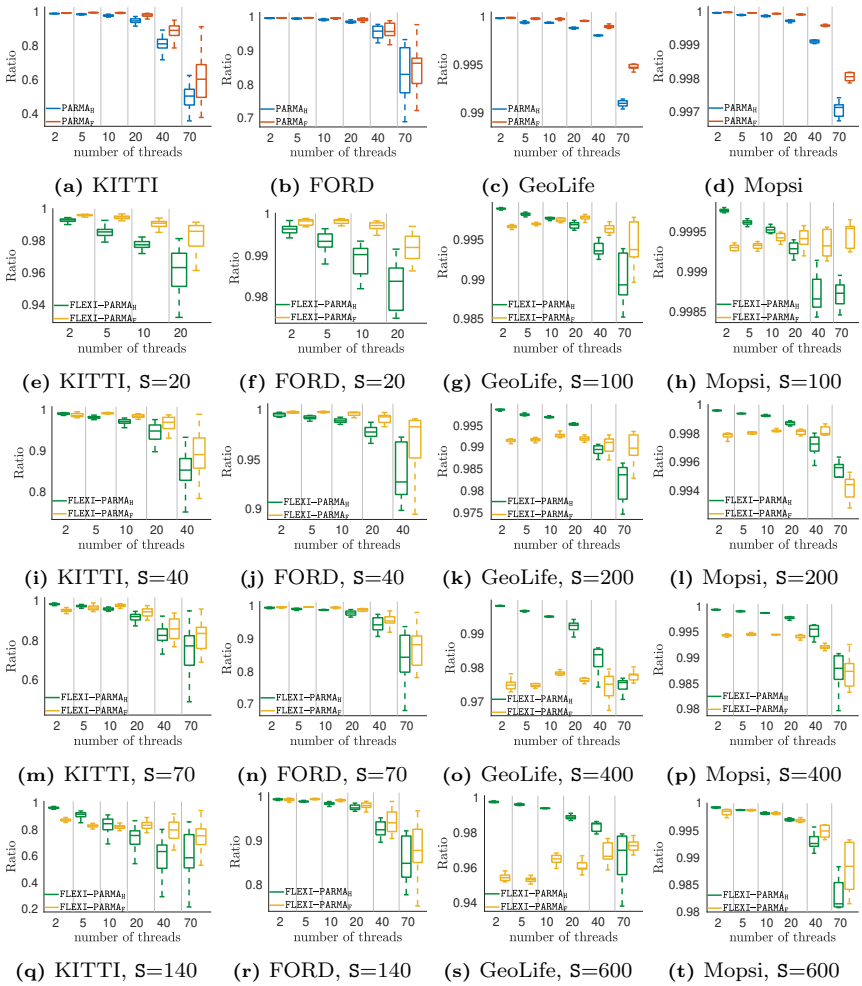
(e) KITTI, **(f)** FORD, **(g)** GeoLife, **(h)** Mopsi, FLEXI-PARMA$_F$ FLEXI-PARMA$_F$ FLEXI-PARMA$_F$ FLEXI-PARMA$_F$

Figure 3.8: *Scalability of* FLEXI-PARMA$_H$ *and* FLEXI-PARMA$_F$ *as a function of* K *and* S *(for* K $\leq$ S*) demonstrated by heat maps for different datasets. Moving between the columns of each heat-map indicates the effect of parallelization, and moving between the rows of each heat-map indicates the effect of approximation.*

The Effects of Parallelization and Approximation on the Scalability of PARMA-CC Algorithms

We have seen so far how PARMA-CC algorithms utilize approximation and parallelization to gain scalability. We here aim to gain insight into the effects of approximation and parallelization on the scalability. To that end, the heat maps in Figure 3.8 summarize the scalability of FLEXI-PARMA$_H$ and FLEXI-PARMA$_F$ for KITTI, FORD, GeoLife, and Mopsi datasets as a function of the number of threads and the number of splits via heat-maps (brighter colours indicate higher scalabilities, and the green parts correspond to the cases in which K is larger

than S). Note that PARMA-CC algorithms yield equivalent clustering results with a fixed value of S (see Corollary 3.1). Therefore, moving between the columns of each heat-map indicates the effect of parallelization (i.e., number of threads), and moving between the rows of each heat-map indicates the effect of approximation.



(a) KITTI **(b)** FORD **(c)** GeoLife **(d)** Mopsi

(e) KITTI, S=20 **(f)** FORD, S=20 **(g)** GeoLife, S=100 **(h)** Mopsi, S=100

(i) KITTI, S=40 **(j)** FORD, S=40 **(k)** GeoLife, S=200 **(l)** Mopsi, S=200

(m) KITTI, S=70 **(n)** FORD, S=70 **(o)** GeoLife, S=400 **(p)** Mopsi, S=400

(q) KITTI, S=140 **(r)** FORD, S=140 **(s)** GeoLife, S=600 **(t)** Mopsi, S=600

Figure 3.9: *Ratio of the duration of longest phase I to the completion time in PARMA-CC algorithms*

3.9.3 Relative Ratio of Local Clustering to the Completion Time

Figure 3.9 shows the ratio of the duration of longest phase I to the completion time in PARMA-CC algorithms. In all cases, with small values for K and S, ratio is very close to one because the local clustering phase constitutes the most significant duration in a PARMA-CC algorithm. Generally, for each dataset, as K or S increases, the aforementioned ratio decreases accordingly.

The latter indicates the presence of two opposing phenomena. Firstly, the local clustering tasks get distributed more evenly among the workers, resulting in higher scalability. On the other hand, as indicated in Observation 3.2, too large values for `K` and/or `S` can increase the expected completion time of phase II, resulting in lower scalability. In § 3.9.2, we empirically studied the joint effects of the aforementioned opposing phenomena on the overall completion time and scalability of PARMA-CC algorithms.

3.9.4 Clustering Accuracy

The right Y-axes in Figure 3.5a and Figure 3.5b show the average accuracy of basic on the KITTI and FORD datasets, respectively. Furthtermore, Figure 3.5c and Figure 3.5d show the accuracy of basic PARMA-CC algorithms on the GeoLife and Mopsi datasets, respectively.

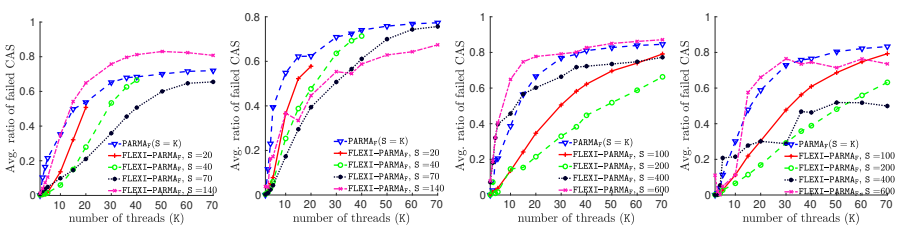
Similarly, the right Y-axes in Fig. 3.5e-3.5t show the clustering accuracy of the flexi PARMA-CC algorithms for varying choices of `S` for each dataset. Note that, in each case, with a fixed value of `S`, PARMA-CC algorithms achieve the same clustering accuracy, as noted in Corollary 3.1.

The right Y-axes in Figure 3.6a and Figure 3.6b show the accuracy of basic PARMA-CC algorithms on the shuffled GeoLife and shuffled Mopsi, respectively.

The right Y-axes in Figure 3.7a, Figure 3.7b, Figure 3.7c, and Figure 3.7d shows the accuracy of basic PARMA-CC algorithms utilizing DBSCAN as the local clustering algorithm on the KITTI, FORD, GeoLife, and Mopsi datasets, respectively.

The results show that, although as `S` increases, the clustering accuracy of PARMA-CC algorithms gradually decreases, in most cases it stays high and this is due to the summarization properties of the bounding ellipsoids. Furthermore, PARMA-CC algorithms that utilize the Euclidean clustering algorithm are able to better keep up the accuracy compared to the PARMA-CC algorithms that utilize DBSCAN.

3.9.5 Shared Memory Contention



(a) KITTI **(b)** FORD **(c)** GeoLife **(d)** Mopsi

Figure 3.10: *The average ratio of failed* `CAS` *operations to the total number of invoked* `CAS` *operations in flat PARMA-CC algorithms (for* $\mathtt{K} \leq \mathtt{S}$*).*

As mentioned in § 3.7.4, there is no shared memory contention in $\mathtt{PARMA_H}$, and the number of occasions in which shared memory contention can take place

in FLEXI-PARMA$_H$ is significantly smaller than that of flat PARMA-CC algorithms. Therefore, we focus on shared memory contention in flat PARMA-CC algorithms. Figure 3.10 shows shared memory contention in those algorithms, as the average ratio of failed CAS operations to the total number of invoked CAS operations for $K \leq S$. Specifically, Figure 3.10a, Figure 3.10b, Figure 3.10c, and Figure 3.10d show the shared memory contention in PARMA$_F$ and FLEXI-PARMA$_F$ on the KITTI, FORD, GeoLife, and Mopsi datasets, respectively. The results show that shared memory contention in PARMA$_F$ is higher than that of FLEXI-PARMA$_F$, with a few exceptions. Furthermore, the results suggest that contention in FLEXI-PARMA$_F$ gets lower by choosing larger values of S, as it increases the number of shared tasks. However, the contention increases again if the chosen number of splits is too large for the amount of data, indicating that too large S values should be avoided for proper use of the algorithms.

3.9.6 Summary of the Empirical Evaluation

We studied the performance and behaviour of PARMA-CC algorithms in a variety of situations. We saw PARMA-CC algorithms achieve significantly higher scalability than the available sequential and parallel algorithms. Notably, we saw super-linear scalability of PARMA-CC algorithms when the dataset is skewed. We also saw that the local clustering is the dominant factor in the execution of a PARMA-CC algorithm. To that end, we noted that the flexi PARMA-CC algorithms improve scalability by increasing S (the number of data splits), up to a point justified by the volume of the data (the splits should not become too small, else the benefits of work-partitioning gets counterbalanced by the overhead to coordinate the latter). Furthermore, with lower inter-split overlap, we observed that the flat PARMA-CC algorithms achieve higher scalability than the hierarchical PARMA-CC algorithms, and we noticed that the hierarchical PARMA-CC algorithms achieve higher scalability when the inter-split overlap is high. We also showd in practice the trade-off between S and the scalability when data is not too big, as well as the clustering accuracy of the algorithms, showing the advantage of suitable choice of S for utilizng the good properties of the PARMA-CC algorithms.

3.10 Related Work

PARMA$_H$, the first parallel multiphase approximate clustering combining algorithm, was introduced and explored in [118]. The present chapter extends the study of PARMA-CC algorithms by considering a design space along two orthogonal aspects. The first aspect considers how the threads synchronize and collaborate, and the second aspect considers how the workload gets distributed among the threads. As a result, the present chapter introduces optimized algorithms targeting different places in the design space. As suggested by the extensive empirical evaluation, different PARMA-CC algorithms can be utilized according to certain properties of the data to be clustered.

In the following, we present three categories of clustering algorithms relevant to PARMA-CC algorithms.

CAT1 The methods in this category can directly be embedded in PARMA-CC algorithms as a local clustering algorithm to gain the scalability benefits of

PARMA-CC algorithms. For example, instead of DBSCAN, DENCLUE [48], STING [49], or OPTICS [50], and their approximate variants (see **CAT3**) can be employed. The algorithms in this category can utilize spatial data structures such as kd-trees [58], Octrees [119], R-trees [120], M-trees [121], and navigating nets [122]. Similarly, PARMA-CC algorithms can also incorporate the utilization of such spatial data structures in the local clustering phase. Moreover, with appropriately formed input, one can also employ Lisco [46], which is a single-pass continuous version of PCL-EC with faster ϵ-neighbourhood radius search via exploiting the angularly sorted readings of a LIDAR sensor.

CAT2 These methods boost the performance of classical clustering algorithms such as DBSCAN through parallelization. For instance, Highly Parallel DBSCAN [64], HPDBSCAN, is an OpenMP/MPI hybrid algorithm that redistributes the points to distinct computational units that perform the local clustering tasks. Then, the local clusters that need to get merged are identified, and thus appropriate cluster relabeling rules get generated, broadcasted, and applied locally. HPDBSCAN offers good scalability; however, when the data is skewed, its performance degrades severely. On the other hand, PARMA-CC algorithms can better tolerate skewed data as shown in the empirical evaluation. Moreover, PARMA-CC algorithms' approach to utilize the shared memory via in-place operations is more efficient that OpenMP's relaxed consistency memory model in which multiple copies of the same data might exist [111]. G-DBSCAN [123] is a parallel version of DBSCAN using GPU that employs a graph structure for indexing data. Other efforts on parallelizing DBSCAN employ a master-slave architecture, e.g., [124]. Nevertheless, PARMA-CC algorithms follow the orthogonal approach of scaling up before scaling out.

CAT3 Methods in this category sacrifice clustering accuracy to gain performance. For example, ρ-approximate DBSCAN [20], and STING (also in **CAT1**) which are both grid-based methods. The former gives a result that is *sandwiched* between those of DBSCAN with parameters $(\epsilon, \texttt{minPts})$ and $(\epsilon(1+\rho), \texttt{minPts})$, for an arbitrary small ρ. With a constant input dimensionality d, ρ-approximate DBSCAN has an expected $\mathcal{O}(\texttt{N})$ complexity. [66]. However, the number of neighbouring cells, $\mathcal{O}(1+(1/\rho)^{d-1})$, grows exponentially with the number of dimensions [20]. STING builds a hierarchical grid structure that divides the spatial area into rectangular cells, at a different resolution per level. Each cell summarizes the points it contains, thus approximating the clustering result of DBSCAN. With a smaller granularity step, the approximation gets better, but the number of bottom layer cells increases. Moreover, same as other grid-based methods, the number of grid cells increase exponentially with the number of input dimensions. Other methods integrate approximate nearest neighbour search techniques (e.g., those based on locality sensitive hashing) into DBSCAN, e.g., [20]. Another approximation approach is to cluster sampled data. To that end, for example, the dynamic (biased) sampling method in [72] can be utilized. The aforementioned techniques can as well be embedded in PARMA-CC algorithms.

3.11 Conclusions

To address the problem of parallel approximate distance- and density-based clustering, we explored a design space for synchronization and workload distribution among the threads. To cover different parts of the design space, we proposed representative PARMA-CC algorithms. We analytically and empirically provided evidence regarding capabilities of PARMA-CC algorithms to balance scaling and accuracy as well as to tolerate skewed data distributions. Furthermore, our studies show that certain properties in the input dataset can determine which PARMA-CC algorithm to choose for the best performance. Moreover, we showed that all PARMA-CC algorithms yield equivalent clustering results. We saw, furthermore, that the approximation technique can result in super-linear scalability in the number of threads, with only marginal loss in accuracy. In general our results show that high-quality approximate clustering can be several orders of magnitude faster than exact clustering. Based on the results of our extensive study of PARMA-CC algorithms, we provide some general guidelines related to parallel approximate data processing in the following:

- Regarding parallelization: In addition to the nature of the data processing task, some intrinsic properties of data also influence the required amount of synchronization among the threads. Fine-grained synchronization techniques are beneficial until certain threshold, but when heavy synchronization is needed, lock-based data parallel approaches can be more efficient. For example, several threads in a flat PARMA-CC algorithm can concurrently merge overlapping ellipsoids in split-summaries. Nonetheless, if the number of overlapping ellipsoids is large (which is a factor determined by the input data), then a large portion of fine-grained synchronization primitives will fail due to contention; consequently, the corresponding threads will need to retry. On the other hand, a thread in a hierarchical PARMA-CC algorithm can merge as many overlapping objects as required without any interruption, while other threads can in parallel merge the ellipsoids in mutually disjoint sets of objects.

- Regarding data structures: The choice of the data structures (and the computational complexity of the required functionalities) should be in accordance with the data processing task. For example, PARMA-CC algorithms utilize a union-set data structure supporting efficient `union` and `find` operations, which is in accordance with the *agglomerative* [125] nature of PARMA-CC algorithms. On the other hand, for a *divisive* [126] data clustering approach, the union-set data structure is probably not a good choice as it does not support efficient separation of sets. From an algorithmic implementation point of view, it is beneficial if the data structures support in-place operations utilizing pointer manipulation techniques.

- Regarding skewed data distributions: Classical data indexing methods used for data clustering can result in quadratic complexity in terms of the size of data under skewed data distributions. Our results show approximation can be a key idea for alleviating the challenges imposed by

high skewness. Furthermore, our study shows splitting a highly skewed data into a number of portions with similar distributions, and performing the required computation on the portions separately and then aggregating the results can reduce the required workload. Despite its approximate nature, our results show such an approach can attain high clustering accuracy.

We expect that PARMA-CC algorithms can facilitate pipeline processing of point clouds, especially combined with stream-processing oriented data structures as proposed in [102, 127] and given the discussion about possible use-cases and associated queries in the respective section. Considering the observed scalability results of PARMA-CC algorithms, a possible future venue of studies and experiments is to adapt PARMA-CC algorithms to GPU enabled systems.

Chapter 4

Parallel Approximate Clustering for High Dimensional Data

IP.LSH.DBSCAN: Integrated Parallel Density-Based Clustering through Locality-Sensitive Hashing

Amir Keramatian, Vincenzo Gulisano, Marina Papatriantafilou and Philippas Tsigas

This Chapter is an adaptation of the article that is submitted for publication.

Summary

Locality-sensitive hashing (LSH) is an established method for fast data indexing and approximate similarity search, with useful parallelism properties. Although indexes and similarity measures are key for data clustering, little has been investigated on the benefits of LSH in the problem. Our proposition is that LSH can be extremely beneficial for parallelizing high-dimensional density-based clustering e.g., DBSCAN, a versatile method able to detect clusters of different shapes and sizes.

We contribute to fill the gap between the advancements in LSH and density-based clustering. In particular, we show how approximate DBSCAN clustering can be *fused* into the process of creating an LSH index structure, and, through data parallelization and fine-grained synchronization, also utilize efficiently available computing capacity as needed for massive data-sets. The resulting method, `IP.LSH.DBSCAN`, can effectively support a wide range of applications with diverse distance functions, as well as data distributions and dimensionality. Furthermore, `IP.LSH.DBSCAN` facilitates adjustable accuracy through LSH parameters. We analyse its properties and also evaluate our prototype implementation on a 36-core machine with 2-way hyper threading on massive data-sets with various numbers of dimensions. Our results show that `IP.LSH.DBSCAN` effectively complements established state-of-the-art methods by up to several orders of magnitude of speed-up on higher dimensional datasets, with tunable high clustering accuracy.

4.1 Introduction

Digitalized applications' datasets are getting larger in size and number of features (i.e., dimensions), posing new challenges to established data mining methods such as data clustering, an unsupervised mining tool based on similarity measures. Density-based spatial clustering of applications with noise (DBSCAN) [47] is a prominent method to cluster (possibly) noisy data into arbitrary shapes and sizes, without prior knowledge on the number of clusters, and using user-defined similarity metrics (i.e., not limited to the Euclidean one). DBSCAN is used in many applications, including LiDAR [6], object detection [128], and GPS route analysis [7]. DBSCAN and some of its variants have been also used to cluster high dimensional data, e.g., medical images [11], text [12], and audio [13].

The computational complexity of traditional DBSCAN is in the worst-case quadratic in the input size [20], expensive considering attributes of today's datasets. Nonetheless, indexing and spatial data structures facilitating proximity searches can ease DBSCAN's computational complexity, as shown with KD-trees [58], R-trees [120], M-trees [121], and cover trees [129]. Using such structures is suboptimal in at least three cases, though: (i) skewed data distributions negatively affect their performance [128], (ii) the *dimensionality curse* results in exact spatial data structures based on deterministic space partitioning being slower than linear scan [130], and (iii) such structures only work for a particular metric (e.g. Euclidean distance). In the literature, the major means for enhancing time-efficiency are those of parallelization [7, 63, 64, 123, 124] and approximation [65], studied alone or jointly [7, 128]. However, state-of-the-art methods target Euclidean distance only and suffer from skewed data distributions and the dimensionality curse.

Locality-sensitive hashing (LSH) is an established approach for approximate similarity search. Based on the idea that if two data points are close using a custom similarity measure, then an appropriate hash function can map them to equal values with high probability [19, 29, 52], LSH can support applications that tolerate *approximate* answers, close to the accurate ones *with high probability*. LSH-based indexing has been successful (and shown to be the best known method [19]) for finding similar items in large high-dimensional data-sets. With our contribution, the `IP.LSH.DBSCAN` algorithm, we show how the processes of approximate density-based clustering and of creating an LSH indexing structure can be *fused* to boost parallel data analysis. Our novel fused approach can efficiently cope with high dimensional data, skewed distributions, large number of points, and a wide range of distance functions. We evaluate the algorithms analytically and empirically, showing they complement the landscape of established state-of-the-art methods, by offering up to several orders of magnitude speed-up on higher dimensional datasets, with tunable high clustering accuracy.

Organization: § 4.2 reviews the preliminaries. § 4.3 and § 4.4 describe and analyse the proposed `IP.LSH.DBSCAN`. § 4.5 covers the empirical evaluation. Related work and conclusions are presented in § 4.6 and § 4.7, respectively.

4.2 Preliminaries

4.2.1 System Model and Problem Description

Let `D` denote an *input* set of `N` points, each a multi-dimensional vector from a domain $\mathcal{D}$, and having a unique `ID`. `Distance` is a distance function applicable on $\mathcal{D}$'s elements. The *goal* is to partition `D` into an a priori unknown number of disjoint clusters, based on `Distance` and parameters `minPts` and ϵ: `minPts` specifies a lower threshold for the number of neighbors, within radius ϵ, for points to be clustered together.

We aim for an efficient, scalable parallel solution, trading approximations in the clustering with reduced calculations regarding the density criteria, while targeting high accuracy. Our evaluation metric for efficiency is *completion time*. Accuracy is measured with respect to an exact baseline using *rand index* [51]: given two clusterings of the same dataset, the *rand index* is the ratio of the number of pairs of elements that are either clustered together or separately in both clusterings, to the total number of pairs of elements. Regarding concurrency guarantees, a common consistency goal is that for every parallel execution, there exists a sequential one producing an equivalent result.

We consider multi-core shared-memory systems executing `K` threads, supporting `read`, `write` and `read-modify-write` atomic operations, e.g. `CAS` (Compare-And-Swap), commonly available in contemporary general purpose processors.

4.2.2 Locality Sensitive Hashing (LSH)

LSH is an indexing technique to facilitate finding similar data points in given a dataset. As mentioned in the introduction, LSH is based on the idea that the hash values (i.e., indexes) of two nearby data points are equal with a high probability given an appropriate choice of locality-sensitive hash functions. The following defines the *sensitivity* of a family of LSH functions [18, 19], i.e., the property that, with high probability, similar points hash to the same value, and dissimilar ones hash to different ones.

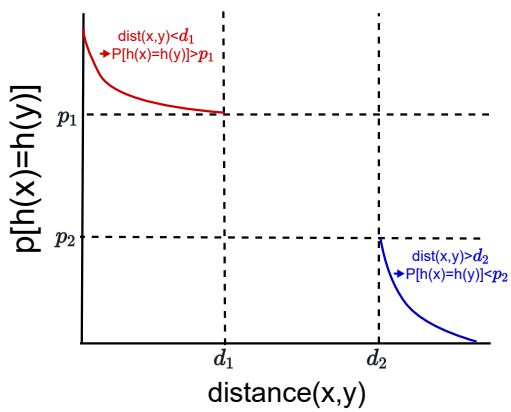


Figure 4.1: Visual illustration of $(\mathtt{d}_1, \mathtt{d}_2, \mathtt{p}_1, \mathtt{p}_2)$-sensitivity for a family of hash functions.

Definition 4.1. *A family of functions* $\mathcal{H} = \{\mathtt{h} : \mathtt{S} \to \mathtt{U}\}$ *is* $(\mathtt{d}_1, \mathtt{d}_2, \mathtt{p}_1, \mathtt{p}_2)$*-sensitive for distance function* `Distance` *if for any* `p` *and* `q` *in* `S` *the following conditions hold: (i) if* $\mathtt{Distance}(\mathtt{p}, \mathtt{q}) \leq d_1$*, then* $\Pr_{\mathcal{H}}[\mathtt{h(p)} = \mathtt{h(q)}] \geq \mathtt{p}_1$ *(ii) if* $\mathtt{Distance}(\mathtt{p}, \mathtt{q}) \geq d_2$*, then* $\Pr_{\mathcal{H}}[\mathtt{h(p)} = \mathtt{h(q)}] \leq \mathtt{p}_2$*. The probabilities are over the random choices in* $\mathcal{H}$*.*

A family $\mathcal{H}$ is useful when $\mathtt{p}_1 > \mathtt{p}_2$ and $\mathtt{d}_1 < \mathtt{d}_2$. Figure 4.1 visualizes the $(\mathtt{d}_1, \mathtt{d}_2, \mathtt{p}_1, \mathtt{p}_2)$-sensitivity for a family of hash functions. LSH functions can be *combined*, into more effective (in terms of sensitivity) ones, as follows [18]:

Definition 4.2. *(i) AND-construction: Given a* $(\mathtt{d}_1, \mathtt{d}_2, \mathtt{p}_1, \mathtt{p}_2)$*-sensitive family* $\mathcal{H}$ *and an integer* `M`*, we can create a new LSH family* $\mathcal{G} = \{\mathtt{g} : \mathtt{S} \to \mathtt{U}^{\mathtt{M}}\}$ *by aggregating/concatenating* `M` *LSH functions from* $\mathcal{H}$*, where* $\mathtt{g(p)}$ *and* $\mathtt{g(q)}$ *are equal iff* $\mathtt{h_j(p)}$ *and* $\mathtt{h_j(q)}$ *are equal for all* $\mathtt{j} \in \{1, \cdots, \mathtt{M}\}$*, implying* $\mathcal{G}$ *is* $(\mathtt{d}_1, \mathtt{d}_2, {\mathtt{p}_1}^{\mathtt{M}}, {\mathtt{p}_2}^{\mathtt{M}})$*-sensitive; (ii) OR-construction: Given an LSH family* $\mathcal{G}$ *and an integer* `L`*, we can create a new LSH family* $\mathcal{F}$ *where each* $\mathtt{f} \in \mathcal{F}$ *consists of* `L` $\mathtt{g_i}$*s chosen independently and uniformly at random from* $\mathcal{G}$*, where* $\mathtt{f(p)}$ *and* $\mathtt{f(q)}$ *are equal iff* $\mathtt{g_j(p)}$ *and* $\mathtt{g_j(q)}$ *are equal for at least one* $\mathtt{j} \in \{1, \cdots, \mathtt{L}\}$*.* $\mathcal{F}$ *is* $(\mathtt{d}_1, \mathtt{d}_2, 1-(1-{\mathtt{p}_1}^{\mathtt{M}})^{\mathtt{L}}, 1-(1-{\mathtt{p}_2}^{\mathtt{M}})^{\mathtt{L}})$*-sensitive assuming* $\mathcal{G}$ *is* $(\mathtt{d}_1, \mathtt{d}_2, {\mathtt{p}_1}^{\mathtt{M}}, {\mathtt{p}_2}^{\mathtt{M}})$*-sensitive.*

LSH structure: An instance of family $\mathcal{F}$ is implemented as `L` hash tables; the `i`-th table is constructed by hashing each point in `D` using $\mathtt{g_i}$ [19, 29]. The resulting data structure associates each *bucket* with the values for the keys mapping to its index. LSH families can associate with various distance functions [18], e.g.:
LSH for Euclidean distance: Let `u` be a randomly chosen unit vector in $\mathcal{D}$. A hash function $\mathtt{h_u(x)}$ in such a family is defined as $\lfloor \frac{\mathtt{x} \cdot \mathtt{u}}{\epsilon} \rfloor$, being $\cdot$ the inner product and ϵ a constant. The family is applicable for any number of dimensions. In a 2-dimensional domain, it is $(\epsilon/2, 2\epsilon, 1/2, 1/3)$-sensitive.
LSH for angular distance: Let `u` be a randomly chosen vector in $\mathcal{D}$. A hash function $\mathtt{h_u(x)}$ in such a family is defined as $\mathtt{sgn}(\mathtt{x} \cdot \mathtt{u})$. The family is $(\theta_1, \theta_2, 1 - \frac{\theta_1}{\pi}, 1 - \frac{\theta_2}{\pi})$-sensitive, where θ_1 and θ_2 are any two angles (in radians) such that $\theta_1 < \theta_2$.

4.2.3 Related Terms and Algorithms

DBSCAN: partitions `D` into an a priori unknown number of clusters, each consisting of at least one *core point* (i.e., one with at least `minPts` points in its ϵ-radius neighbourhood) and the points that are *density-reachable* from it. Point q is density-reachable from p, if q is *directly reachable* from p (i.e., in its ϵ-radius neighbourhood) or from another core point that is density-reachable from p. Non-core points that are density-reachable from some core-point are called *border points*, while others are noise [20]. DBSCAN can utilize any distance function e.g., Euclidean, Jaccard, Hamming, angular [18]. Its worst-case time complexity is $\mathcal{O}(\mathtt{N}^2)$, but in certain cases (e.g. for Euclidean distance and low-dimensional datasets) its expected complexity lowers to $\mathcal{O}(\mathtt{N} \log \mathtt{N})$, through indexing structures facilitating range queries to find ϵ neighbours [20].
HP-DBSCAN [64]: Highly Parallel DBSCAN is an OpenMP/MPI algorithm, super-imposing a hyper-grid over the input set. It distributes the points to computing units that do local clusterings. Then, the local clusters that need

merging are identified and cluster relabeling rules get broadcasted and applied locally.

PDS-DBSCAN [63]: An exact parallel version of Euclidean DBSCAN that uses a spatial indexing structure for efficient query ranges. It parallelizes the work by partitioning the points and merging partial clusters, maintained via a *disjoint-set* data structure, also known as *union-find* (a collection of disjoint sets, with the elements in each set connected as a directed tree). Such a data structure facilitates *in-place* `find` and `merge` operations [109] avoiding data copying. Given an element p, `find` retrieves the root (i.e., the *representative*) of the tree in which p resides, while `merge` merges the sets containing two given elements.

Theoretically-Efficient and Practical Parallel DBSCAN [7]: Via a grid-based approach, this algorithm identifies core-cells and utilizes a union-find data structure to merge the neighbouring cells having points within ϵ-radius. It uses spatial indexes to facilitate finding neighbourhood cells and answering range queries.

LSH as index for DBSCAN: LSH's potential led other works (e.g., [70, 71]) to consider it as a plain means for neighbourhood queries. We refer to them as `VLSHDBSCAN`.

4.3 The Proposed IP.LSH.DBSCAN Method

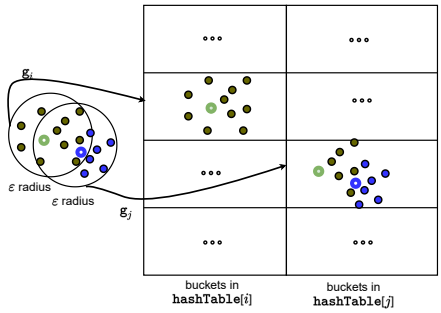
`IP.LSH.DBSCAN` utilizes the LSH properties, for parallel density-clustering, through efficient fusion of the indexing and clustering formation.

At a high level, `IP.LSH.DBSCAN` hashes each point in `D`, into multiple hash-tables, in such a way that with a high probability, points within ϵ-distance get hashed to the same bucket at least once across all the tables. E.g., Figure 4.2a shows how most nearby points in a subset of `D` get hashed to the same buckets, in two hash tables. Subsequently, the buckets containing at least `mintPts` elements are examined, to find a set of *candidate core-points* which later will be filtered to identify the real *core-points*, in terms of DBSCAN's definition. In Figure 4.2a, the core-points are shown as bold points with a dot inside. The buckets containing core points are characterized as *core buckets*. Afterwards, with the help of the hash tables, core-points within each others' ϵ-neighbourhood get merged. E.g., the core-bucket in the rightmost hash table in Figure 4.2a contains two core-points, indicating the possibility that they are within each other's ϵ-neighbourhood, in which case they get merged. The merging is done using a forest of union-find data structures, consisting of such core-points, that essentially represent core buckets. As we see later, multiple threads can work in parallel in these steps.

4.3.1 Key Elements and Phases

Similar to an LSH structure (cf. § 4.2.2), we utilize `L` hash tables (`hashTable`[1], $\cdots$, `hashTable`[`L`]), each constructed using `M` hash functions, chosen according to distance metric `Distance` and threshold ϵ (see § 4.2.2, § 4.4).

Definition 4.3. *A bucket in any of the hash tables is called a* candidate core-bucket *if it contains at least* `minPts` *elements. A* candidate core-point `c` *in a*



(a)

N	≜	number of points in the input dataset
d	≜	number of data point dimensions
K	≜	number of threads
L	≜	number of hash tables
M	≜	number of hash functions per table
C	≜	number of core-points identified by LSHDBSCAN

(b) Table of Notation

Figure 4.2: 4.2a shows nearby points get hashed to the same bucket at least once across hash tables, whp. Core-points are the bold ones with a dot inside.

candidate core-bucket `ccb` *is defined to be the closest (using function* `Distance`*) point in* `ccb` *to the centroid of all the points in* `ccb`*; we also say that* `c` represents `ccb`*. A candidate core bucket* `ccb`*, whose candidate core-point* `c` *has at least* `minPts` *points within its* ϵ*-radius in* `ccb`*, is called a* core-bucket. Core-forest *is a concurrent union-find structure containing core-points representing core buckets.*

Lemma 4.1. *A candidate core-point, having in its bucket at least* `minPts` *points within its* ϵ*-radius, is a* core-point *according to the DBSCAN definition (§ 4.2.3).*

The above follows from Definition 4.3. Next we present `IP.LSH.DBSCAN`'s basic phases, followed by detailed description of parallelization and pseudo-code.

In phase I (*hashing and bucketing*), for each i, each point `p` in `D` is hashed using the LSH function $\mathtt{g}_i$ and inserted in `hashTable`[`i`]. Furthermore, the algorithm keeps track of the buckets containing at least `minPts`, as candidate core buckets. In phase II (*core-point identification*), for each candidate core-bucket, the algorithm identifies a candidate core-point. If at least `minPts` points in a candidate core-bucket fall within the ϵ-neighbourhood of the identified candidate core-point, the latter is identified as a *true core-point* and inserted into the core-forest as a singleton. In phase III (*merge-task identification and processing*), the algorithm inspects each core-bucket and creates and performs a *merge task* for each pair of core-points that are within each others' ϵ-neighbourhood. Hence, the elements in the core-forest start forming sets according to the merge tasks. In phase IV (*data labeling*), the algorithm labels the points: a core point gets assigned the same clustering label as all the other core points with which it forms a set in the core-forest. A border point (i.e., a non-core point located in the ϵ-radius of a core-point) is labeled the same as a corresponding core-point, and all the other points are considered noise.

4.3.2 Parallelism and Algorithmic Implementation

We here present the parallelization in `IP.LSH.DBSCAN` (see Alg. 4.1), targeting speed-up by distributing the workload among `K` threads. We also aim at in-place operations on data points and buckets (i.e., without creating additional copies),

hence work with pointers to the relevant data points and buckets in the data structures.

Algorithm 4.1 Outline of IP.LSH.DBSCAN

```
1: Input: dataset D, threshold minPts, radius ε, nr. of hash tables L, nr. of hash functions per
   table M, metric Distance, nr of threads K; Output: a clustering label for each point in D
2: let D be logically partitioned into S mutually disjoint batches
3: hashTable[1],···,hashTable[L] are hash tables supporting concurrent insertions and traversals
4: candidateCoreBuckets and coreBuckets are empty sets supporting concurrent operations
5: let hashTasks be a S × L boolean array initialized to false, indicating the status of hash tasks
   corresponding to the Cartesian product of S batches and L hash tables
6: let G = {g : S → U^M} be an LSH family suitable for metric Distance, and let g_1, ···, g_L be
   hash functions chosen independently and uniformly at random from G (Definition 4.2)
7: for all threads in parallel do
8:   phase I: hashing and bucketing
9:     while the running thread can book a task from hashTasks do
10:      for each point p in task.batch do
11:        let i be index of the hashTable associated with task
12:        hashTable[i].insert(key = g_i(p), value = ptr(p))
13:        bucket=hashTable[i].getBucket(key = g_i(p))
14:        if bucket.size() ≥ minPts then candidateCoreBuckets.insert(ptr(bucket))
15:  phase II: core-point identification (starts when all threads reach here)
16:    for each ccb in candidateCoreBuckets do
17:      let c be the closest point in ccb to ccb points' centroid
18:      if |{q ∈ ccb such that Distance(c, q)}| ≥ minPts then
19:        c→ corePoint := TRUE and insert c into the core-forest
20:        coreBuckets.insert(ccb)
21:  phase III: merge-task identification and processing (starts when all threads reach here)
22:    while cb := coreBuckets.pop() do
23:      let core be the core-point associated with cb
24:      for core-point c ∈ cb such that Distance(core, c) ≤ ε do merge(core, c)
25:  phase IV: data labeling (starts when a thread reaches here)
26:    for each core bucket cb do
27:      let core be the core-point associated with cb
28:      for each non-labeled point p in cb do
29:        if p→ corePoint then p.idx = findRoot(p).ID
30:        else p.idx = findRoot(core).ID
```

Phase I (*hashing and bucketing*): The first step is to parallelize the hashing of the input dataset D into L hash tables. We (logically) partition D into S mutually disjoint batches. Consecutively, we have S × L *hash tasks*, corresponding to the Cartesian product of the hash tables and the data batches. We utilize a mechanism through which the threads can *book* a hash task and thus share the workload. To that end, hashTasks is a boolean S × L array containing a *status* for each task, initially false. A thread in phase I scans the elements of hashTasks, and if it finds an non-booked task, then it tries to *atomically book* the task (e.g. via a CAS operation to change the status from false to true). The thread that successfully books a hash task $\texttt{ht}_{\texttt{b},\texttt{t}}$ hashes each data point p in batch b into hashTable[t] using hash function $\texttt{g}_\texttt{t}$. Particularly, for each point p, a key-value pair consisting of the hashed value of p and a pointer to p is inserted in hashTable[t]. As entries get inserted into the hash tables, pointers to buckets with at least minPts points are stored in the set candidateCoreBuckets. Since threads concurrently operate on the same tables, we use hash tables supporting concurrent insertions and traversals [112]. Alg. 4.1 l.8-l.14 summarizes Phase I.

Phase II (*core-point identification*): Here the threads identify core-buckets and core-points. Each thread atomically pops a candidate core bucket ccb from candidateCoreBuckets. Afterwards, it identifies the closest point to the

centroid of the points in ccb, considering it as a candidate core-point, ccp. If there are at least minPts points in ccb within ϵ-radius of ccp, then ccp and ccb are identified as core-point and core-bucket, respectively, and ccp is inserted in the core-forest and the ccb in the coreBucekts set. This phase, shown in Alg. 4.1 l.15-l.20, is finished when candidateCoreBuckets becomes empty.

Phase III (*merge-task identification and processing*): The threads here identify and perform merge tasks. For each core-bucket cb that a thread successfully books from the set coreBuckets, the thread merges the sets corresponding to the associated core-point with cb and any other core-point in cb within ϵ distance. For merging, the algorithm uses an established concurrent implementation for disjoint-sets, with *linearizable* and *wait-free* (i.e., the effects of concurrent operations appear instantaneously and are consistent with the sequential specification, while the threads can make progress independently of each other [107]) find and merge, proposed in [107]. The phase is finished when coreBucekts becomes empty. Its steps are shown in Alg. 4.1 l.21-l.24.

Phase IV (*data labeling*): Each non-labeled core-point in a core-bucket gets assigned its root ID in the core-forest as the clustering label. The clustering label assigned to all other non-labeled points in a core-bucket is the root ID of the associated core-point. The aforementioned process, shown in Alg. 4.1 l.25-l.30, can be performed concurrently for all the core-buckets.

4.4 Analysis

We study the consistency, time and memory properties of IP.LSH.DBSCAN. Figure 4.2b summarizes the notations.

At the end of phase IV, each set in the core-forest contains a subset of density-reachable core-points (as defined in § 4.2). Two disjoint-set structures ds_1, ds_2 are *equivalent* if there is a one-to-one correspondence between ds_1's and ds_2's sets. The following lemma implies that the outcomes of single-threaded and concurrent executions of IP.LSH.DBSCAN are equivalent.

Lemma 4.2. *Any pair of concurrent executions of* IP.LSH.DBSCAN *that use the same hash functions, result to equivalent core-forests at the end of phase IV.*

Proof sketch. Considering a fixed instance of the problem, any concurrent execution of IP.LSH.DBSCAN identifies the same set of core-points and core-buckets with the same hash functions, hence performing the same set of merge operations. As the concurrent executions of merge operations are linearizable (see § 4.3) and merge operation satisfies the associative and commutative properties, the resulting sets in the core-forest are identical for any concurrent execution.

It is worth noting that *border points* (i.e., non-core points within the vicinity of multiple core-points) can be assigned to any of the neighbouring clusters. The original DBSCAN [47] exhibits the same behaviour as well. Let C be the size of the core-forest, i.e., number of identified core-points by IP.LSH.DBSCAN.

Lemma 4.3. *[adapted from Theorem 2 in [109]] The probability that each* findRoot *and each* merge *perform* $\mathcal{O}(\log \mathtt{C})$ *steps is at least* $1 - \frac{1}{\mathtt{C}}$.

Corollary 4.1. *The expected asymptotic time complexity of each* `findRoot` *and each* `merge` *is* $\mathcal{O}(\log \mathtt{C})$.

Lemma 4.4. *The expected completion time of phase I is* $\mathcal{O}(\frac{\mathtt{LMNd}}{\mathtt{K}})$*; phase II and phase III is bounded by* $\mathcal{O}(\frac{\mathtt{LN}\log \mathtt{C}}{\mathtt{K}})$*; phase IV is* $\mathcal{O}(\frac{\mathtt{N}\log \mathtt{C}}{\mathtt{K}})$.

Proof sketch. In the following, we argue about each of the listed items. We take into consideration that each insertion into a hash table or a set, as associative containers, takes constant time with respect to `L`, `M`, and `d`.

- Phase I. The dominant workload in this phase is to hash `N` `d`-dimensional data points into `L` hash tables using `M` functions.
- Phase II. To identify the candidate core points, all the `N` points are iterated in each table in the worst-case. Similar is the worst case for identifying the merge tasks. For most instances of the problem, the expected completion time of phase II and III can be significantly smaller than the worst-case bound.
- Phase III. A `merge` operation (whose expected time complexity is given in Corollary 4.1) is performed for each identified merge task, the latter being in the worst case linear in the number of buckets. This upper bound is loose for most data distributions as in the previous case.
- Phase IV. In this phase, a maximum of `N` `findRoot` operations, each with expected time complexity of $\mathcal{O}(\log \mathtt{C})$ (Corollary 4.1), are performed.

The partitioning into `S` batches and the fine-grained work-sharing schemes as described in the previous section make it possible for the work-load to get evenly distributed among the `K` threads in each of the above cases.

Theorem 4.1. *The expected completion time of* `IP.LSH.DBSCAN` *is* $\mathcal{O}(\frac{\mathtt{LMNd}+\mathtt{LN}\log \mathtt{C}}{\mathtt{K}})$.

Theorem 4.1 is derived according to the asymptotically dominant terms in Lemma 4.4. Theorem 4.1 shows `IP.LSH.DBSCAN`'s expected completion time is inversely proportional to `K` and grows linearly in `N`, `d`, `L`, and `M`. In common cases where `C` is much smaller than `N`, the expected completion time of `IP.LSH.DBSCAN` is $\mathcal{O}(\frac{\mathtt{LMNd}}{\mathtt{K}})$. In the worst-case, where `C` is $\mathcal{O}(\mathtt{N})$, the expected completion time is $\mathcal{O}(\frac{\mathtt{LMNd}+\mathtt{LN}\log \mathtt{N}}{\mathtt{K}})$. For this to happen, for instance, ϵ and `minPts` need to be extremely small and `L` be extremely large. As the density parameters of DBSCAN are chosen to detect meaningful clusters, such choices for ϵ and `minPts` are in practice avoided.

On the memory use of `IP.LSH.DBSCAN`: The memory footprint of `IP.LSH.DBSCAN` is proportional to $(\mathtt{LN} + \mathtt{Nd})$, as it simply needs only one copy of each data point and pointers in the hash tables and this dominates the overhead of all other utilized data structures. Further, in-place operations ensure that data is not copied and transferred unnecessarily, which is a significant factor regarding efficiency. In § 4.5, the effect of these properties is discussed.

Choice of `L` and `M`: For an LSH structure, a plot representing the probability of points hashing into the same bucket as a function of their distance resembles an inverse s-curve (x- and y-axis being the distance, and the probability of

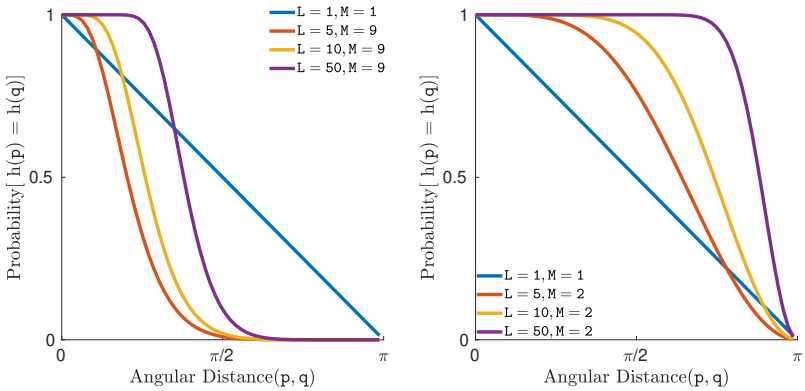


Figure 4.3: The probability of two given points p and q being hashed to the same bucket as a function of their angular distance using a variety of cascading, where the AND construction is applied first, and then cascaded with an OR construction.

hashing to the same bucket, resp.), starting at 1 for the points with distance 0, declining with a significant slope around some threshold, and approaching 0 for far apart points. For example, Figure 4.3 shows the inverse s-curves corresponding to different choices of L and M for the angular distance. As shown in Figure 4.3, the choice of L and M directly influence the shape of the associated curve, particularly the location of the threshold and the sharpness of the decline [18]. It is worth noting that steeper declines generally result in more accurate LSH structures at the expense of larger L and M values. Consequently, in IP.LSH.DBSCAN, L and M must be determined to (i) set the location of the threshold at ϵ, and (ii) balance the trade-off between the steepness of the decline and the completion time. In § 4.5, we study a range of L and M values and their implications on the trade-off between IP.LSH.DBSCAN's accuracy and completion time.

4.5 Evaluation

We conduct an extensive evaluation of IP.LSH.DBSCAN, comparing it with the established state-of-the-art algorithms. Our implementation is publicly available [131]. Complementing Theorem 4.1, we measure the execution latency with varying number of threads (K), data points (N), dimensions (d), hash tables (L), and hash functions per table (M). We use varying ϵ values, as well as Euclidean and angular distances. We measure IP.LSH.DBSCAN's accuracy against the exact DBSCAN (hence also the baseline state-of-the-art algorithms) using rand index.

Setup: We implemented IP.LSH.DBSCAN in C++, using POSIX threads and the concurrent hash table of Intel's threading building blocks [112] library (TBB). We used a c5.18xlarg AWS machine, with 144 GB of memory and two Intel Xeon Platinum 8124M CPUs, with 36 two-way hyper-threaded cores [7] in total.

Tested Methods: In addition to IP.LSH.DBSCAN, we perform experiments with PDSDBSCAN [63], HPDBSCAN [64], and the exact algorithm in [7], for which we use the label TEDBSCAN (Theoretically-Efficient and Practical Parallel

DBSCAN). As the approximate algorithms in [7] are generally not faster than their exact counterpart (see Fig. 9 and discussion on p. 13 in [7]), we consider their efficiency represented by the exact TEDBSCAN. We also implemented and tested `VLSHDBSCAN`, our version of a single-thread DBSCAN that uses LSH indexing, as we did not find open implementations for [70, 71]. Benchmarking `VLSHDBSCAN` allows a comparison regarding the approximation degree, as well the efficiency induced by the "fused" approach `IP.LSH.DBSCAN` that leads to the efficient combination of searching and combining through the same hash table. The aforementioned algorithms are reviewed in § 4.2.3.

4.5.1 Evaluation Data & Parameters

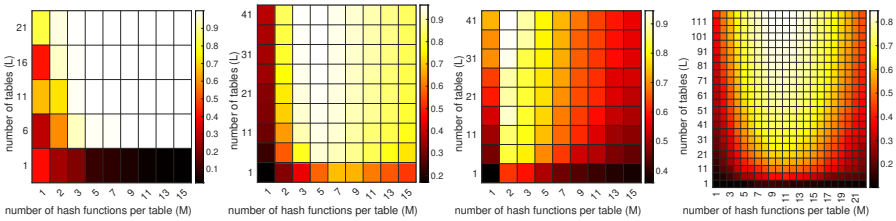
Following common practices [7, 64, 132], we use datasets with different characteristics. We use varying ϵ but fixed `minPts`, as the sensitivity on the latter is significantly smaller [132]. We also follow earlier works' common practice to abort any execution that exceeds a certain bound, here $9\texttt{X}10^5$ seconds (more than 24 hours). We introduce the datasets and the chosen values for ϵ and `minPts` as well as the choices for `L` and `M`, based on the corresponding discussion in § 4.4 and also the literature guidelines (e.g., [18] and the reference therein). The *default* ϵ values are shown in *italics*.

TeraClickLog [7]: Each point in this dataset corresponds a display ad served by Criteo and contains 13 integer and 26 categorical features. We choose a subset containing over 67 million points, free from missing features. Similar to [7], we consider the 13 integer features, and we choose ϵ from $\{\textit{1500}, 3000, 6000, 12000\}$ and `minPts` 100. Figure 4.4a visualizes the rand index accuracy of `IP.LSH.DBSCAN` on our subset of the TeraClickLog dataset as a function of `L` and `M` via a heat-map for $\epsilon = 1500$. We choose {`L`=5, `M`=5}, {`L`=10, `M`=5}, and {`L`=20, `M`=5} giving 0.98, 0.99, and 1 rand index accuracy, respectively.

Household [65]: This is an electricity consumption dataset with over two million points, each being seven-dimensional after removing the date and time features (as suggested in [65]). Following the practice in [7, 65], we scale each feature to [0,10000] interval and choose ϵ from $\{1500, \textit{2000}, 2500, 3000\}$ and `minPts` 100. Figure 4.4b visualizes the rand index accuracy of `IP.LSH.DBSCAN` on the Household dataset as a function of `L` and `M` via a heat-map for $\epsilon = 2000$. We choose {`L`=5, `M`=5}, {`L`=10, `M`=5}, and {`L`=20, `M`=5} giving 0.92, 0.94, and 0.95 rand index accuracy, respectively.

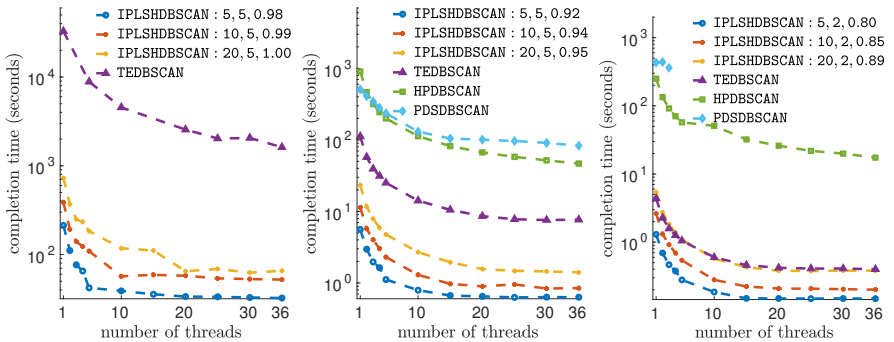
GeoLife [133]: This is a GPS trajectory dataset. We used approximately 1.45 million points as selected in [128], containing latitude and longitude with a *highly skewed* distribution. Similar to [128], we choose ϵ from $\{\textit{0.001}, 0.002, 0.004, 0.008\}$ and `minPts` 500. Figure 4.4c visualizes the rand index accuracy of `IP.LSH.DBSCAN` on our subset of the GeoLife dataset as a function of `L` and `M` via a heat-map for $\epsilon = 0.001$. We choose {`L`=5, `M`=2}, {`L`=10, `M`=2}, and {`L`=20, `M`=2} giving 0.8, 0.85, and 0.89 rand index accuracy, respectively.

MNIST: The set contains 70000 records, each of them a 28`X`28-pixel handwritten digit 0-9, where the actual labels are available [40]. We treat each record as a 784-dimensional data point, and normalized each data point to have a unit length (similar to [31]). We utilize the angular distance. Following [134], we choose ϵ from $\{0.18\pi, 0.19\pi, \textit{0.20}\pi, 0.21\pi\}$ and `minPts` 100. Figure 4.4d visualizes the rand index accuracy of `IP.LSH.DBSCAN` on the MNIST dataset



(a) TeraClickLog ϵ=1500, `minPts`=100 **(b)** Household ϵ=2000, `minPts`=100 **(c)** GeoLife ϵ=0.001, `minPts`=500 **(d)** MNIST $\epsilon = 0.2\pi$, `minPts`=100

Figure 4.4: *Heat-maps visualizing the rand index accuracy of* `IP.LSH.DBSCAN` *as a function of* `L` *and* `M`



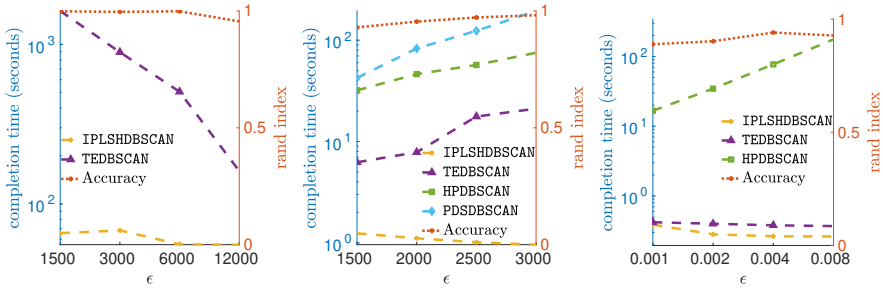
(a) TeraClickLog ϵ:1500, `minPts`:100 $\|$`VLSHDBSCAN`: 5, 5, -$\| > 9X10^5$ **(b)** Household ϵ:2000, `minPts`:100 $\|$`VLSHDBSCAN`: 5, 5, 0.99$\|$=1666 **(c)** Geolife ϵ:0.001, `minPts`:500 $\|$`VLSHDBSCAN`: 5, 2, 0.99$\|$=1671

Figure 4.5: *Completion time with varying* `K`*. The comma-separated values corresponding to* `IP.LSH.DBSCAN` *and* `VLSHDBSCAN` *show* `L`*,* `M`*, and the rand index accuracy, respectively. PDSDBSCAN crashes by running out of memory in 4.5a for all* `K` *and for* `K`≥ 4 *in 4.5c. In 4.5a no HPDBSCAN executions terminate within the* $9X10^5$*-sec threshold.*

as a function of `L` and `M` with respect to the actual labels via a heat-map for $\epsilon = 0.2\pi$. We choose {`L`=58, `M`=9}, {`L`=116, `M`=9}, and {`L`=230, `M`=9} giving 0.77, 0.85, and 0.89 rand index, respectively.

4.5.2 Experiments for the Euclidean Distance

Completion time with varying `K`: Figure 4.5a, Figure 4.5b, and Figure 4.5c show the completion time of `IP.LSH.DBSCAN` and other tested methods with varying `K` on TeraClickLog, Household, and GeoLife datasets, respectively. PDSDBSCAN crashes by running out of memory on TeraClickLog for all `K` and on GeoLife for `K`≥ 4, and none of HPDBSCAN's executions terminate within the $9X10^5$ sec threshold. For the reference, in Figure 4.5, the completion time of single-thread `VLSHDBSCAN` is provided as a caption for each dataset, except for TeraClickLog, for the above reason. The results indicate the benefits of parallelization for work-load distribution in `IP.LSH.DBSCAN`, also validating that `IP.LSH.DBSCAN`'s completion time exhibits a linear behaviour with respect to `L`, as shown in Theorem 4.1. In cases where dimensionality is higher, challenging

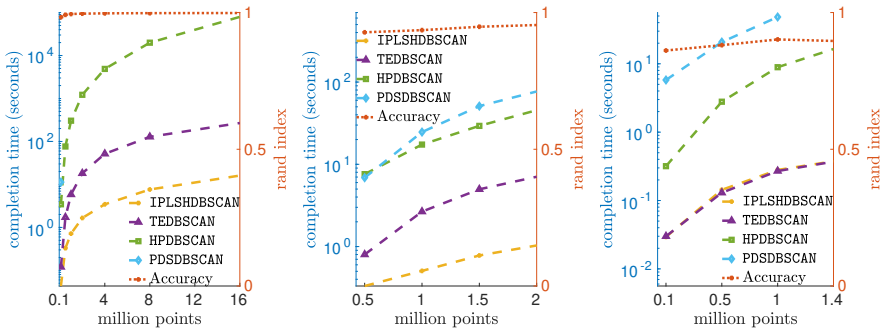


(a) TeraClickLog
`minPts`:100, `L`:20, `M`:5

(b) Household
`minPts`:100, `L`:20, `M`:5

(c) Geolife
`minPts`:500, `L`:20, `M`:2

Figure 4.6: *Completion time using varying ϵ with 36 cores. PDSDBSCAN crashes by running out of memory in 4.6a and 4.6c for all ϵ. None of HPDBSCAN's executions terminate within the $9X10^5$ sec threshold in 4.6a. Right Y-axes show* `IP.LSH.DBSCAN`*'s rand index.*

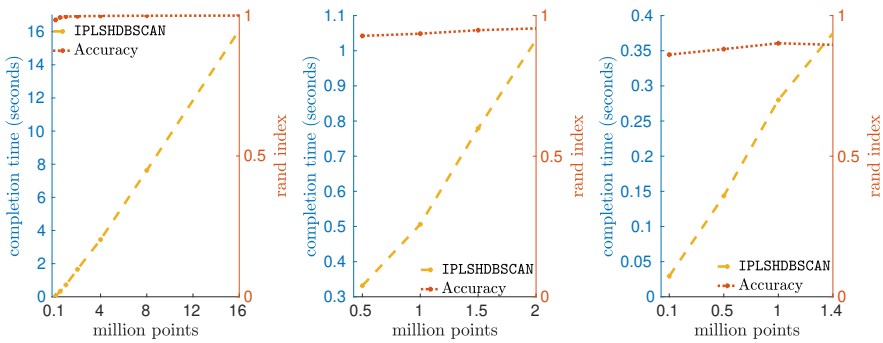


(a) TeraClickLog
ϵ:1500, `minPts`:100, `L`:20, `M`:5

(b) Household
ϵ:2000, `minPts`:100, `L`:20, `M`:5

(c) Geolife
ϵ:0.001, `minPts`:500, `L`:20, `M`:2

Figure 4.7: *Completion time with varying* `N` *using 36 cores. PDSDBSCAN runs out of memory in 4.7a with* `N`*>0.1 million points and with* `N`*>1 million points in 4.7c.* `IP.LSH.DBSCAN` *and TEDBSCAN coincide in 4.7c. Right Y-axes show* `IP.LSH.DBSCAN`*'s rand index.*

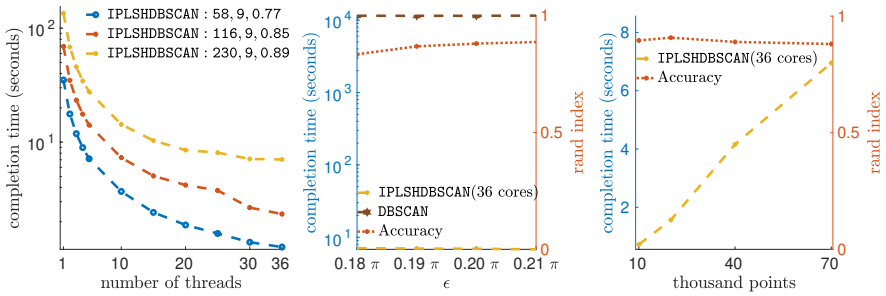


(a) TeraClickLog
`L`:20, `M`:5

(b) Household
`L`:20, `M`:5

(c) Geolife
`L`:20, `M`:2

Figure 4.8: *4.8a, 4.8b, and 4.8c show in linear Y-axes the completion time measurements of* `IP.LSH.DBSCAN` *shown in 4.7a, 4.7b, and 4.7c, respectively. The results show completion time of* `IP.LSH.DBSCAN` *linearly grows with respect to* `N`*.*



(a) $\epsilon : 0.2\pi$, minPts:100 ||VLSHDBSCAN: 58,9,0.55||=6316 **(b)** minPts:100, L:230, M:9 **(c)** $\epsilon : 0.2\pi$, minPts:100, L:230, M:9

Figure 4.9: *MNIST results with the angular distance (only* IP.LSH.DBSCAN, VLSHDBSCAN, *DBSCAN support the angular distance). 4.9a shows* IP.LSH.DBSCAN*'s completion time with varying* K*. The left Y-axes in 4.9b and 4.9c respectively show* IP.LSH.DBSCAN*'s completion time with varying* ϵ *and* N*, using 36 cores. The right Y-axes in 4.9b and 4.9c show the associated accuracy, computed with respect to the actual labels.*

the state-of-the-art algorithms, IP.LSH.DBSCAN's completion time is several orders of magnitude faster.

Completion time with varying ϵ: The left Y-axes in Figure 4.6a, Figure 4.6b, and Figure 4.6c show the completion time of IP.LSH.DBSCAN and other tested methods using 36 cores with varying ϵ values on TeraClickLog, Household, and GeoLife datasets, respectively. PDSDBSCAN crashes by running out of memory on TeraClickLog and GeoLife for all ϵ, and none of HPDBSCAN's executions terminate within the $9X10^5$ sec threshold. The right Y-axes in Figure 4.6a, Figure 4.6b, and Figure 4.6c show the corresponding rand index accuracy of IP.LSH.DBSCAN. The results show that in general the completion time of IP.LSH.DBSCAN decreases by increasing ϵ. Intuitively, hashing points into larger buckets results in lower `merge` workload. Similar benefits, although with higher completion times, are seen for TEDBSCAN. On the other hand, as the results show, completion time of many classical methods (such as HPDBSCAN and PDSDBSCAN) increases with increasing ϵ.

Completion time with varying N: The left Y-axes in Figure 4.7a, Figure 4.7b, and Figure 4.7c show the completion time of the bench-marked methods using 36 cores on varying size subsets of TeraClickLog, Household, and GeoLife datasets, respectively. PDSDBSCAN runs out of memory on TeraClickLog subsets with N>0.1 million points and GeoLife subsets with N>1 million points. For better visibility, the left Y-axes in Figure 4.8a, Figure 4.8b, and Figure 4.8c show only the completion time of IP.LSH.DBSCAN using 36 cores on varying size subsets of TeraClickLog, Household, and GeoLife datasets with linear scale, respectively. The results empirically validate that completion time of IP.LSH.DBSCAN exhibits a linear growth in the number of data points, complementing Theorem 4.1 (complementing figures in provided in [131]). The right Y-axes in Figure 4.7a, Figure 4.7b, and Figure 4.7c show the corresponding rand index accuracy of IP.LSH.DBSCAN.

4.5.3 Experiments for the Angular Distance

For data with significantly high number of dimensions, as a side-effect of dimensionality curse, the Euclidean distance among all pairs of points are almost equal [18]. To overcome this issue, we use the angular distance. We only study the behaviour of IP.LSH.DBSCAN, VLSHDBSCAN, and DBSCAN as the other bench-marked methods do not support the angular distance. Here accuracy is calculated against the actual labels. Figure 4.9a shows IP.LSH.DBSCAN's completion time with varying K. The left Y-axes in Figure 4.9b and Figure 4.9c respectively show IP.LSH.DBSCAN's completion time with varying ϵ and N, using 36 cores. The right Y-axes in Figure 4.9b and Figure 4.9c show the associated accuracies. The results show IP.LSH.DBSCAN's completion time is more than 4 orders of magnitude faster than a sequential DBSCAN and more than 3 orders of magnitude faster than VLSHDBSCAN. Here, too the results align and complement Theorem 4.1's analysis.

4.5.4 Highlights of the Results

IP.LSH.DBSCAN targets high dimensional clustering, in a memory-efficient way, and it supports various distance measures. IP.LSH.DBSCAN's completion time for high-dimensional datasets is several orders of magnitude faster than state-of-the-art counterparts, while ensuring approximation with tunable accuracy and showing efficiency also with lower dimension data as well. In practice, IP.LSH.DBSCAN's completion time exhibits a linear behaviour with respect to the number of points, even for skewed data distributions and varying density parameters. The benefits of IP.LSH.DBSCAN with respect to other algorithms increase with increasing data dimensionality. IP.LSH.DBSCAN scales both with the size of the input and its dimensionality.

4.6 Other Related Work

Density clustering is discussed in many related works. In § 4.5, we compared IP.LSH.DBSCAN with representative state-of-the-art related algorithms. We focus here on the related work considering approximation.

Gan et al. and Wang et al. in [7, 65] proposed approximate DBSCAN grid-based algorithms, targeting only low-dimensional Euclidean distance, but its expected complexity is $\mathcal{O}(\mathtt{N}^2)$ if $2^{\mathtt{d}} > \mathtt{N}$ [135]. Quoting from [20] about grid-based methods: "*Similar to R*-trees and most index structures, grid-based approaches tend to not scale well with increasing number of dimensions, due to the curse of dimensionality [citations]: the number of possible grid cells grows exponentially with the dimensionality*". Intuitively, if the number of dimensions is considered a constant, the corresponding part of the induced overhead does not show in the asymptotic complexity. PARMA-CC [128] is another approximate concurrent clustering algorithm suitable only for low-dimensional data.

VLSHDBSCAN [70, 71] uses LSH for neighbourhood queries. On the other hand, in IP.LSH.DBSCAN, creating the LSH index is embedded into the dynamics of the formation of the clusters. The IP.LSH.DBSCAN iterates over buckets, and it apply merges on core points that represent bigger entities, drastically reducing

the search complexity. Furthermore, IP.LSH.DBSCAN is a concurrent algorithm as opposed to VLSHDBSCAN which is a single-thread algorithm. Esfandiari et al. [13] propose an almost linear approximate DBSCAN that identifies core-points by mapping points into hyper-cubes and counting the points in each hyper-cube. It uses LSH to find and merge nearby core-points. IP.LSH.DBSCAN integrates core-point identification and merging in one structure altogether, leading to better efficiency and flexibility in leveraging the desired distance function.

4.7 Conclusions

In the landscape of algorithmic implementations of DBSCAN, IP.LSH.DBSCAN proposes a simple and efficient method combining insights on DBSCAN with features of LSH. It offers approximation with tunable accuracy and high parallelism, avoiding the exponential growth of the search effort with the number of data dimensions, thus scaling both with the size of the input and its dimensionality, and dealing with high skewness in a memory-efficient way. We expect that the method will help a variety of applications in the evolving landscape of cyberphysical system problems, that require to extract information from very large, high-dimensional, highly-skewed data sets. We also expect that this methodology can be used for partitioning data for other types of graph processing and as such this direction is worth investigating as extension of IP.LSH.DBSCAN.

Bibliography

[1] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aaa8415

[2] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*. Elsevier Science, 2008. [Online]. Available: https://books.google.se/books?id=QgD-3Tcj8DkC

[3] D. Z. Chen, M. H. M. Smid, and B. Xu, "Geometric algorithms for density-based data clustering," *Int. J. Comput. Geom. Appl.*, vol. 15, no. 3, pp. 239–260, 2005. [Online]. Available: https://doi.org/10.1142/S0218195905001683

[4] J. Wu, H. Xu, J. Zheng, and J. Zhao, "Automatic vehicle detection with roadside lidar data under rainy and snowy conditions," *IEEE Intell. Transp. Syst. Mag.*, vol. 13, no. 1, pp. 197–209, 2021. [Online]. Available: https://doi.org/10.1109/MITS.2019.2926362

[5] R. B. Rusu, "Semantic 3d object maps for everyday manipulation in human living environments," *KI - Künstliche Intelligenz*, vol. 24, no. 4, pp. 345–348, Nov 2010. [Online]. Available: https://doi.org/10.1007/s13218-010-0059-6

[6] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (PCL)," in *IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, 9-13 May 2011*. IEEE, 2011. [Online]. Available: https://doi.org/10.1109/ICRA.2011.5980567

[7] Y. Wang, Y. Gu, and J. Shun, "Theoretically-efficient and practical parallel DBSCAN," in *2020 SIGMOD Int. Conf. on Management of Data*. ACM, 2020, pp. 2555–2571. [Online]. Available: https://doi.org/10.1145/3318464.3380582

[8] R. Mariescu-Istodor and P. Fränti, "Grid-based method for GPS route analysis for retrieval," *ACM Trans. Spatial Algorithms Syst.*, vol. 3, no. 3, pp. 8:1–8:28, 2017. [Online]. Available: https://doi.org/10.1145/3125634

[9] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W.-Y. Ma, "Understanding mobility based on gps data," in *Proceedings of the 10th International Conference on Ubiquitous Computing*, ser. UbiComp '08. New York, NY, USA: ACM, 2008, pp. 312–321. [Online]. Available: http://doi.acm.org/10.1145/1409635.1409677

[10] J. Shen, X. Hao, Z. Liang, Y. Liu, W. Wang, and L. Shao, "Real-time superpixel segmentation by DBSCAN clustering algorithm," *IEEE Trans. Image Process.*, vol. 25, no. 12, pp. 5933–5942, 2016. [Online]. Available: https://doi.org/10.1109/TIP.2016.2616302

[11] F. Baselice, L. Coppolino, S. D'Antonio, G. Ferraioli, and L. Sgaglione, "A DBSCAN based approach for jointly segment and classify brain MR images," in *37th Annual Int. Conf. of the IEEE Engineering in Medicine and Biology Society, EMBC 2015*. IEEE, 2015, pp. 2993–2996. [Online]. Available: https://doi.org/10.1109/EMBC.2015.7319021

[12] X. Wang, L. Zhang, X. Zhang, and K. Xie, "Application of improved DBSCAN clustering algorithm on industrial fault text data," in *18th IEEE Int. Conf. on Industrial Inf., INDIN*. IEEE, 2020, pp. 461–468. [Online]. Available: https://doi.org/10.1109/INDIN45582.2020.9442093

[13] H. Esfandiari, V. S. Mirrokni, and P. Zhong, "Almost linear time density level set estimation via DBSCAN," in *Thirty-Fifth AAAI Conf. on Artificial Intelligence, AAAI 2021*. AAAI Press, 2021, pp. 7349–7357. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/16902

[14] M. Chiang and T. Zhang, "Fog and IoT: An Overview of Research Opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, Dec. 2016.

[15] V. Kartashevskiy and M. Buranova, "Analysis of packet jitter in multi-service network," in *2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S T)*, 2018, pp. 797–802.

[16] P. X. Liu, M. Q. Meng, and S. X. Yang, "Data communications for internet robots," *Auton. Robots*, vol. 15, no. 3, pp. 213–223, 2003. [Online]. Available: https://doi.org/10.1023/A:1026160302776

[17] G. Cormode, M. N. Garofalakis, P. J. Haas, and C. Jermaine, "Synopses for massive data: Samples, histograms, wavelets, sketches," *Found. Trends Databases*, vol. 4, no. 1-3, pp. 1–294, 2012. [Online]. Available: https://doi.org/10.1561/1900000004

[18] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of Massive Datasets, 2nd Ed.* Cambridge University Press, 2014. [Online]. Available: http://www.mmds.org/

[19] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proc. of the 30th Annual ACM Symp. on the Theory of Comp.,*. ACM, 1998, pp. 604–613. [Online]. Available: https://doi.org/10.1145/276698.276876

[20] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Dbscan revisited, revisited: Why and how you should (still) use dbscan," *ACM Trans. Database Syst.*, vol. 42, no. 3, pp. 19:1–19:21, Jul. 2017. [Online]. Available: http://doi.acm.org/10.1145/3068335

[21] C. H. Goh, A. Lim, B. C. Ooi, and K. Tan, "Efficient indexing of high-dimensional data through dimensionality reduction," *Data Knowl. Eng.*, vol. 32, no. 2, pp. 115–130, 2000. [Online]. Available: https://doi.org/10.1016/S0169-023X(99)00031-2

[22] H. Lin, "High index compression without the dependencies of data orders and data skewness for spatial databases," *J. Inf. Sci. Eng.*, vol. 27, no. 2, pp. 561–576, 2011. [Online]. Available: http://www.iis.sinica.edu.tw/page/jise/2011/201103_11.html

[23] Y. He, H. Tan, W. Luo, S. Feng, and J. Fan, "MR-DBSCAN: a scalable mapreduce-based DBSCAN algorithm for heavily skewed data," *Frontiers Comput. Sci.*, vol. 8, no. 1, pp. 83–99, 2014. [Online]. Available: https://doi.org/10.1007/s11704-013-3158-3

[24] P. B. Gibbons, "Big data: Scale down, scale up, scale out," in *2015 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2015, Hyderabad, India, May 25-29, 2015*. IEEE Computer Society, 2015, p. 3. [Online]. Available: https://doi.org/10.1109/IPDPS.2015.123

[25] M. H. ur Rehman, C. S. Liew, A. Abbas, P. P. Jayaraman, T. Y. Wah, and S. U. Khan, "Big data reduction methods: A survey," *Data Science and Engineering*, vol. 1, no. 4, pp. 265–284, Dec 2016. [Online]. Available: https://doi.org/10.1007/s41019-016-0022-0

[26] M. M. Gaber, A. B. Zaslavsky, and S. Krishnaswamy, "Mining data streams: a review," *SIGMOD Record*, vol. 34, no. 2, pp. 18–26, 2005. [Online]. Available: https://doi.org/10.1145/1083784.1083789

[27] C. O. S. Sorzano, J. Vargas, and A. D. Pascual-Montano, "A survey of dimensionality reduction techniques," *CoRR*, vol. abs/1403.2877, 2014. [Online]. Available: http://arxiv.org/abs/1403.2877

[28] B. Havers, R. Duvignau, H. Najdataei, V. Gulisano, A. C. Koppisetty, and M. Papatriantafilou, "DRIVEN: a framework for efficient data retrieval and clustering in vehicular networks," in *35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019*. IEEE, 2019, pp. 1850–1861. [Online]. Available: https://doi.org/10.1109/ICDE.2019.00201

[29] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proc. of the 20th Symp. on Comp. Geometry*, ser. SCG '04. ACM, 2004, pp. 253–262. [Online]. Available: http://doi.acm.org/10.1145/997817.997857

[30] P. Kumar, A. Gangal, S. Kumari, and S. Tiwari, "Recombinant sort: N-dimensional cartesian spaced algorithm designed from synergetic combination of hashing, bucket, counting and radix sort," *CoRR*, vol. abs/2107.01391, 2021. [Online]. Available: https://arxiv.org/abs/2107.01391

[31] N. Sundaram, A. Turmukhametova, N. Satish, T. Mostak, P. Indyk, S. Madden, and P. Dubey, "Streaming similarity search over one

billion tweets using parallel locality-sensitive hashing," *Proc. VLDB Endow.*, vol. 6, no. 14, pp. 1930–1941, 2013. [Online]. Available: http://www.vldb.org/pvldb/vol6/p1930-sundaram.pdf

[32] A. Sodani, R. Gramunt, J. Corbal, H. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y. Liu, "Knights landing: Second-generation intel xeon phi product," *IEEE Micro*, vol. 36, no. 2, pp. 34–46, 2016. [Online]. Available: https://doi.org/10.1109/MM.2016.25

[33] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *American Federation of Information Processing Societies: Proceedings of the AFIPS '67 Spring Joint Computer Conference, April 18-20, 1967, Atlantic City, New Jersey, USA*, ser. AFIPS Conference Proceedings, vol. 30. AFIPS / ACM / Thomson Book Company, Washington D.C., 1967, pp. 483–485. [Online]. Available: https://doi.org/10.1145/1465482.1465560

[34] N. J. Gunther, "A general theory of computational scalability based on rational functions," *CoRR*, vol. abs/0808.1431, 2008. [Online]. Available: http://arxiv.org/abs/0808.1431

[35] B. Schwarz, "Practical scalability analysis with the universal scalability law," 2015.

[36] N. Khan, A. Naim, M. R. Hussain, N. N. Quadri, N. Ahmad, and S. Qamar, "The 51 v's of big data: Survey, technologies, characteristics, opportunities, issues and challenges," in *Proceedings of the International Conference on Omni-Layer Intelligent Systems, COINS 2019, Crete, Greece, May 5-7, 2019*. ACM, 2019, pp. 19–24. [Online]. Available: https://doi.org/10.1145/3312614.3312623

[37] R. Appuswamy, C. Gkantsidis, D. Narayanan, O. Hodson, and A. I. T. Rowstron, "Scale-up vs scale-out for hadoop: time to rethink?" in *ACM Symposium on Cloud Computing, SOCC '13, Santa Clara, CA, USA, October 1-3, 2013*. ACM, 2013, pp. 20:1–20:13. [Online]. Available: https://doi.org/10.1145/2523616.2523629

[38] G. Atanacio-Jimenez, J.-J. Gonzalez-Barbosa, J. B. Hurtado-Ramos, F. J. Ornelas-Rodriguez, H. Jimenez-Hernandez, T. Garcia-Ramirez, and R. Gonzalez-Barbosa, "Lidar velodyne hdl-64e calibration using pattern planes," *International Journal of Advanced Robotic Systems*, vol. 8, no. 5, p. 59, 2011. [Online]. Available: https://doi.org/10.5772/50900

[39] B. Schwarz, "Lidar: Mapping the world in 3d," *Nature Photonics*, vol. 4, no. 7, p. 429, 2010.

[40] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. [Online]. Available: https://doi.org/10.1109/5.726791

[41] S. Chen, B. Ma, and K. Zhang, "On the similarity metric and the distance metric," *Theor. Comput. Sci.*, vol. 410, no. 24-25, pp. 2365–2376, 2009. [Online]. Available: https://doi.org/10.1016/j.tcs.2009.02.023

[42] H. Kriegel, M. Schubert, and A. Zimek, "Angle-based outlier detection in high-dimensional data," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*. ACM, 2008, pp. 444–452. [Online]. Available: https://doi.org/10.1145/1401890.1401946

[43] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA, 1967, pp. 281–297.

[44] U. von Luxburg, "A tutorial on spectral clustering," *Stat. Comput.*, vol. 17, no. 4, pp. 395–416, 2007. [Online]. Available: https://doi.org/10.1007/s11222-007-9033-z

[45] D. Yan, L. Huang, and M. I. Jordan, "Fast approximate spectral clustering," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*. ACM, 2009, pp. 907–916. [Online]. Available: https://doi.org/10.1145/1557019.1557118

[46] H. Najdataei, Y. Nikolakopoulos, V. Gulisano, and M. Papatriantafilou, "Continuous and parallel lidar point-cloud clustering," in *38th IEEE International Conference on Distributed Computing Systems, ICDCS 2018, Vienna, Austria, July 2-6, 2018*. IEEE Computer Society, 2018, pp. 671–684. [Online]. Available: https://doi.org/10.1109/ICDCS.2018.00071

[47] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *2nd Conf. on Knowledge Discovery and Data Mining (KDD-96)*. AAAI Press, 1996, pp. 226–231. [Online]. Available: http://www.aaai.org/Library/KDD/1996/kdd96-037.php

[48] A. Hinneburg and D. A. Keim, "An efficient approach to clustering in large multimedia databases with noise," in *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98), New York City, New York, USA, August 27-31, 1998*. AAAI Press, 1998, pp. 58–65. [Online]. Available: http://www.aaai.org/Library/KDD/1998/kdd98-009.php

[49] W. Wang, J. Yang, and R. R. Muntz, "Sting: A statistical information grid approach to spatial data mining," in *Proceedings of the 23rd International Conference on Very Large Data Bases*, ser. VLDB '97. Morgan Kaufmann Publishers Inc., 1997, pp. 186–195. [Online]. Available: http://dl.acm.org/citation.cfm?id=645923.758369

[50] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," in *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '99. New York, NY, USA: ACM, 1999, pp. 49–60. [Online]. Available: http://doi.acm.org/10.1145/304182.304187

[51] S. Wagner and D. Wagner, "Comparing clusterings - an overview," Universität Karlsruhe (TH), Tech. Rep. 4, 2007.

[52] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Commun. ACM*, vol. 51, no. 1, pp. 117–122, 2008. [Online]. Available: https://doi.org/10.1145/1327452.1327494

[53] M. Charikar, "Similarity estimation techniques from rounding algorithms," in *34th ACM Symp. on Theory of Comp.* ACM, 2002, pp. 380–388. [Online]. Available: https://doi.org/10.1145/509907.509965

[54] N. Elmqvist and P. Tsigas, "A taxonomy of 3d occlusion management for visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 5, pp. 1095–1109, Sep. 2008. [Online]. Available: https://doi.org/10.1109/TVCG.2008.59

[55] J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz, and E. G. Steinbach, "Real-time compression of point cloud streams," in *IEEE International Conference on Robotics and Automation, ICRA 2012, 14-18 May, 2012, St. Paul, Minnesota, USA*. IEEE, 2012, pp. 778–785. [Online]. Available: https://doi.org/10.1109/ICRA.2012.6224647

[56] M. Chiang and T. Zhang, "Fog and iot: An overview of research opportunities," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 854–864, 2016. [Online]. Available: https://doi.org/10.1109/JIOT.2016.2584538

[57] O. Akribopoulos, I. Chatzigiannakis, C. Tselios, and A. Antoniou, "On the deployment of healthcare applications over fog computing infrastructure," in *41st IEEE Annual Computer Software and Applications Conference, COMPSAC 2017, Turin, Italy, July 4-8, 2017. Volume 2*. IEEE Computer Society, 2017, pp. 288–293. [Online]. Available: https://doi.org/10.1109/COMPSAC.2017.178

[58] J. L. Bentley, "K-d trees for semidynamic point sets," in *6th Symp. on Comp. Geometry*. ACM, 1990, pp. 187–197. [Online]. Available: https://doi.org/10.1145/98524.98564

[59] J. Elseberg, D. Borrmann, and A. Nüchter, "One billion points in the cloud – an octree for efficient processing of 3d laser scans," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 76, pp. 76–88, 2013, terrestrial 3D modelling. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271612001888

[60] W. Wang, J. Yang, and R. Muntz, *PK-Tree: A Spatial Index Structure for High Dimensional Point Data*. Boston, MA: Springer US, 2000, pp. 281–293. [Online]. Available: https://doi.org/10.1007/978-1-4615-1379-7_20

[61] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, "On the surprising behavior of distance metrics in high dimensional spaces," in *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings*, ser. Lecture Notes in Computer Science, vol. 1973. Springer, 2001, pp. 420–434. [Online]. Available: https://doi.org/10.1007/3-540-44503-X_27

[62] A. Flexer and D. Schnitzer, "Choosing l_p norms in high-dimensional spaces based on hub analysis," *Neurocomputing*, vol. 169, pp. 281–287, 2015. [Online]. Available: https://doi.org/10.1016/j.neucom.2014.11.084

[63] M. M. A. Patwary, D. Palsetia, A. Agrawal, W. Liao, F. Manne, and A. N. Choudhary, "A new scalable parallel DBSCAN algorithm using the disjoint-set data structure," in *SC Conf. on High Perf. Comp. Networking, Storage and Analysis, SC '12*. IEEE/ACM, 2012, p. 62. [Online]. Available: https://doi.org/10.1109/SC.2012.9

[64] M. Götz, C. Bodenstein, and M. Riedel, "HPDBSCAN: highly parallel DBSCAN," in *Workshop on Machine Learning in High-Perf. Comp. Environments, MLHPC 2015*. ACM, 2015, pp. 2:1–2:10. [Online]. Available: https://doi.org/10.1145/2834892.2834894

[65] J. Gan and Y. Tao, "On the hardness and approximation of euclidean DBSCAN," *ACM Trans. Database Syst.*, vol. 42, no. 3, pp. 14:1–14:45, 2017. [Online]. Available: https://doi.org/10.1145/3083897

[66] ——, "DBSCAN revisited: Mis-claim, un-fixability, and approximation," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*. ACM, 2015, pp. 519–530. [Online]. Available: https://doi.org/10.1145/2723372.2737792

[67] M. Sualeh and G. Kim, "Dynamic multi-lidar based multiple object detection and tracking," *Sensors*, vol. 19, no. 6, p. 1474, 2019. [Online]. Available: https://doi.org/10.3390/s19061474

[68] E. Januzaj, H.-P. Kriegel, and M. Pfeifle, "Towards effective and efficient distributed clustering," in *In Workshop on Clustering Large Data Sets (ICDM)*, 2003, pp. 49–58.

[69] G. Forman and B. Zhang, "Distributed data clustering can be efficient and exact," *SIGKDD Explorations*, vol. 2, no. 2, pp. 34–38, 2000. [Online]. Available: https://doi.org/10.1145/380995.381010

[70] Y.-P. Wu, J.-J. Guo, and X.-J. Zhang, "A linear dbscan algorithm based on lsh," in *Int. Conf. on ML and Cybernetics*, vol. 5, 2007, pp. 2608–2614.

[71] Y. Shiqiu and Z. Qingsheng, "Dbscan clustering algorithm based on locality sensitive hashing," *Journal of Physics: Conf. Series*, vol. 1314, p. 012177, 10 2019.

[72] G. Kollios, D. Gunopulos, N. Koudas, and S. Berchtold, "Efficient biased sampling for approximate clustering and outlier detection in large data sets," *IEEE Trans. on Knowl. and Data Eng.*, vol. 15, no. 5, pp. 1170–1187, Sep. 2003. [Online]. Available: https://doi.org/10.1109/TKDE.2003.1232271

[73] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe LSH: efficient indexing for high-dimensional similarity search," in *Proceedings of the 33rd International Conference on*

Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007. ACM, 2007, pp. 950–961. [Online]. Available: http://www.vldb.org/conf/2007/papers/research/p950-lv.pdf

[74] ——, "Intelligent probing for locality sensitive hashing: Multi-probe LSH and beyond," *Proc. VLDB Endow.*, vol. 10, no. 12, pp. 2021–2024, 2017. [Online]. Available: http://www.vldb.org/pvldb/vol10/p2021-lv.pdf

[75] D. Göhring, M. Wang, M. Schnürmacher, and T. Ganjineh, "Radar/lidar sensor fusion for car-following on highways," in *5th International Conference on Automation, Robotics and Applications, ICARA 2011, Wellington, New Zealand, December 6-8, 2011*. IEEE, 2011, pp. 407–412. [Online]. Available: https://doi.org/10.1109/ICARA.2011.6144918

[76] S. Agarwal, A. Vora, G. Pandey, W. Williams, H. Kourous, and J. R. McBride, "Ford Multi-AV Seasonal Dataset," *CoRR*, vol. abs/2003.07969, 2020. [Online]. Available: https://arxiv.org/abs/2003.07969

[77] S. Kumari, P. Goyal, A. Sood, D. Kumar, S. Balasubramaniam, and N. Goyal, "Exact, fast and scalable parallel dbscan for commodity platforms," in *18th Conf. on Distributed Comp. and Networking*, ser. ICDCN '17, 2017, pp. 14:1–14:10. [Online]. Available: http://doi.acm.org/10.1145/3007748.3007773

[78] A. Keramatian, V. Gulisano, M. Papatriantafilou, P. Tsigas, and Y. Nikolakopoulos, "MAD-C: multi-stage approximate distributed cluster-combining for obstacle detection and localization," in *Euro-Par 2018: Parallel Processing Workshops - Euro-Par 2018 International Workshops, Turin, Italy, August 27-28, 2018, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 11339. Springer, 2018, pp. 312–324. [Online]. Available: https://doi.org/10.1007/978-3-030-10549-5_25

[79] M. Himmelsbach, F. von Hundelshausen, and H. Wünsche, "Fast segmentation of 3d point clouds for ground vehicles," in *IEEE Intelligent Vehicles Symposium (IV), 2010, La Jolla, CA, USA, June 21-24, 2010*. IEEE, 2010, pp. 560–565. [Online]. Available: https://doi.org/10.1109/IVS.2010.5548059

[80] A. S. Tanenbaum and M. van Steen, *Distributed systems - principles and paradigms, 2nd Edition*. Pearson Education, 2007.

[81] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975. [Online]. Available: http://doi.acm.org/10.1145/361002.361007

[82] N. Hansen, "The CMA evolution strategy: A tutorial," *CoRR*, vol. abs/1604.00772, 2016. [Online]. Available: http://arxiv.org/abs/1604.00772

[83] S. Alfano and M. Greer, "Determining if two solid ellipsoids intersect," *Journal of Guidance Control and Dynamics - J GUID CONTROL DYNAM*, vol. 26, pp. 106–110, 01 2003.

[84] D. M. W. Powers, "Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation," *CoRR*, vol. abs/2010.16061, 2020. [Online]. Available: https://arxiv.org/abs/2010.16061

[85] N. M. Preguiça, J. M. Marquès, M. Shapiro, and M. Letia, "A commutative replicated data type for cooperative editing," in *29th IEEE International Conference on Distributed Computing Systems (ICDCS 2009), 22-26 June 2009, Montreal, Québec, Canada*. IEEE Computer Society, 2009, pp. 395–403. [Online]. Available: https://doi.org/10.1109/ICDCS.2009.20

[86] E. G. Zimbelman, R. F. Keefe, E. K. Strand, C. A. Kolden, and A. M. Wempe, "Hazards in motion: Development of mobile geofences for use in logging safety," *Sensors*, vol. 17, no. 4, p. 822, 2017. [Online]. Available: https://doi.org/10.3390/s17040822

[87] M. Galassi, J. Davies, J. Theiler, B. Gough, and G. Jungman, *GNU Scientific Library - Reference Manual, Third Edition, for GSL Version 1.12*. Network Theory Ltd, 2009. [Online]. Available: http://www.network-theory.co.uk/gsl/manual/

[88] C. Kohlhoff. (2019) Boost.Asio. [Online]. Available: https://www.boost.org/doc/libs/1_66_0/doc/html/boost_asio.html

[89] V. Oracle. (2019) Virtualbox. [Online]. Available: https://www.virtualbox.org/

[90] O. Michel, "Webots: Professional mobile robot simulation," *CoRR*, vol. abs/cs/0412052, 2004. [Online]. Available: http://arxiv.org/abs/cs/0412052

[91] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *I. J. Robotics Res.*, vol. 32, no. 11, pp. 1231–1237, 2013. [Online]. Available: https://doi.org/10.1177/0278364913491297

[92] A. Nguyen and B. Le, "3d point cloud segmentation: A survey," in *IEEE 6th International Conference on Robotics, Automation and Mechatronics, RAM 2013, Manila, Philippines, November 12-15, 2013*, 2013, pp. 225–230. [Online]. Available: https://doi.org/10.1109/RAM.2013.6758588

[93] J. M. Cebrian, B. Imbernón, J. A. Soto, J. M. García, and J. M. Cecilia, "High-throughput fuzzy clustering on heterogeneous architectures," *Future Gener. Comput. Syst.*, vol. 106, pp. 401–411, 2020. [Online]. Available: https://doi.org/10.1016/j.future.2020.01.022

[94] Y. Djenouri, D. Djenouri, A. Belhadi, and A. Cano, "Exploiting GPU and cluster parallelism in single scan frequent itemset mining," *Inf. Sci.*, vol. 496, pp. 363–377, 2019. [Online]. Available: https://doi.org/10.1016/j.ins.2018.07.020

[95] P. Eisert, E. G. Steinbach, and B. Girod, "Multi-hypothesis, volumetric reconstruction of 3-d objects from multiple calibrated camera views," in *Proceedings of the 1999 IEEE International Conference on Acoustics,*

Speech, and Signal Processing, ICASSP '99, Phoenix, Arizona, USA, March 15-19, 1999. IEEE Computer Society, 1999, pp. 3509–3512. [Online]. Available: https://doi.org/10.1109/ICASSP.1999.757599

[96] M. N. Garofalakis, J. Gehrke, and R. Rastogi, Eds., *Data Stream Management - Processing High-Speed Data Streams*, ser. Data-Centric Systems and Applications. Springer, 2016. [Online]. Available: https://doi.org/10.1007/978-3-540-28608-0

[97] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*. IEEE Computer Society, 2012, pp. 3354–3361. [Online]. Available: https://doi.org/10.1109/CVPR.2012.6248074

[98] K. Huebner, S. Ruthotto, and D. Kragic, "Minimum volume bounding box decomposition for shape approximation in robot grasping," in *2008 IEEE International Conference on Robotics and Automation, ICRA 2008, May 19-23, 2008, Pasadena, California, USA*, 2008, pp. 1628–1633. [Online]. Available: https://doi.org/10.1109/ROBOT.2008.4543434

[99] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," in *3rd International Conference on 3D Digital Imaging and Modeling (3DIM 2001), 28 May - 1 June 2001, Quebec City, Canada*. IEEE Computer Society, 2001, pp. 145–152. [Online]. Available: https://doi.org/10.1109/IM.2001.924423

[100] M. Y. Ansari, A. Ahmad, S. S. Khan, G. Bhushan, and Mainuddin, "Spatiotemporal clustering: a review," *Artif. Intell. Rev.*, vol. 53, no. 4, pp. 2381–2423, 2020. [Online]. Available: https://doi.org/10.1007/s10462-019-09736-1

[101] Z. Fu, M. Almgren, O. Landsiedel, and M. Papatriantafilou, "Online temporal-spatial analysis for detection of critical events in cyber-physical systems," in *2014 IEEE International Conference on Big Data (IEEE BigData 2014), Washington, DC, USA, October 27-30, 2014*. IEEE Computer Society, 2014, pp. 129–134. [Online]. Available: https://doi.org/10.1109/BigData.2014.7004221

[102] V. Gulisano, Y. Nikolakopoulos, I. Walulya, M. Papatriantafilou, and P. Tsigas, "Deterministic real-time analytics of geospatial data streams through scalegate objects," in *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 316–317. [Online]. Available: https://doi.org/10.1145/2675743.2776758

[103] N. Richerzhagen, R. Kluge, B. Richerzhagen, P. Lieser, B. Koldehofe, I. Stavrakakis, and R. Steinmetz, "Better together: Collaborative monitoring for location-based services," in *19th IEEE International Symposium on "A World of Wireless, Mobile and Multimedia Networks", WoWMoM 2018, Chania, Greece, June 12-15, 2018*. IEEE Computer Society, 2018, pp. 14–22. [Online]. Available: https://doi.org/10.1109/WoWMoM.2018.8449798

[104] A. Keramatian, V. Gulisano, M. Papatriantafilou, and P. Tsigas, "MAD-C: multi-stage approximate distributed cluster-combining for obstacle detection and localization," *J. Parallel Distributed Comput.*, vol. 147, pp. 248–267, 2021. [Online]. Available: https://doi.org/10.1016/j.jpdc.2020.08.013

[105] C. L. Glennie and D. D. Lichti, "Static calibration and analysis of the velodyne HDL-64E S2 for high accuracy mobile scanning," *Remote. Sens.*, vol. 2, no. 6, pp. 1610–1624, 2010. [Online]. Available: https://doi.org/10.3390/rs2061610

[106] M. Keuper, E. Levinkov, N. Bonneel, G. Lavoué, T. Brox, and B. Andres, "Efficient decomposition of image and mesh graphs by lifted multicuts," in *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 2015, pp. 1751–1759. [Online]. Available: https://doi.org/10.1109/ICCV.2015.204

[107] M. Herlihy, "Wait-free synchronization," *ACM Trans. Program. Lang. Syst.*, vol. 13, no. 1, p. 124–149, Jan. 1991. [Online]. Available: https://doi.org/10.1145/114005.102808

[108] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. [Online]. Available: http://mitpress.mit.edu/books/introduction-algorithms

[109] S. V. Jayanti and R. E. Tarjan, "A randomized concurrent algorithm for disjoint set union," in *2016 ACM Symp. on Principles of Distributed Comp.* ACM, 2016. [Online]. Available: https://doi.org/10.1145/2933057.2933108

[110] R. D. Blumofe and C. E. Leiserson, "Scheduling multithreaded computations by work stealing," *J. ACM*, vol. 46, no. 5, pp. 720–748, 1999. [Online]. Available: https://doi.org/10.1145/324133.324234

[111] J. P. Hoeflinger and B. R. de Supinski, "The openmp memory model," in *OpenMP Shared Memory Parallel Programming - International Workshops, IWOMP 2005 and IWOMP 2006, Eugene, OR, USA, June 1-4, 2005, Reims, France, June 12-15, 2006. Proceedings*, ser. Lecture Notes in Computer Science, vol. 4315. Springer, 2005, pp. 167–177. [Online]. Available: https://doi.org/10.1007/978-3-540-68555-5_14

[112] T. Willhalm and N. Popovici, "Putting intel threading building blocks to work," in *1st Int. Workshop on Multicore Software Eng.*, ser. IWMSE '08. ACM, 2008, p. 3–4. [Online]. Available: https://doi.org/10.1145/1370082.1370085

[113] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou, "Cilk: An efficient multithreaded runtime system," *J. Parallel Distributed Comput.*, vol. 37, no. 1, pp. 55–69, 1996. [Online]. Available: https://doi.org/10.1006/jpdc.1996.0107

[114] S. B. Pope, "Algorithms for ellipsoids," *Cornell Univ., Rep. No. FDA*, 2008.

[115] G. van den Bergen, "Efficient collision detection of complex deformable models using AABB trees," *J. Graphics, GPU, & Game Tools*, vol. 2, no. 4, pp. 1–13, 1997. [Online]. Available: https://doi.org/10.1080/10867651.1997.10487480

[116] S. Gottschalk, M. C. Lin, and D. Manocha, "Obbtree: A hierarchical structure for rapid interference detection," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1996, New Orleans, LA, USA, August 4-9, 1996*. ACM, 1996, pp. 171–180. [Online]. Available: https://doi.org/10.1145/237170.237244

[117] Y. Zheng, L. Zhang, X. Xie, and W. Ma, "Mining interesting locations and travel sequences from GPS trajectories," in *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*. ACM, 2009, pp. 791–800. [Online]. Available: https://doi.org/10.1145/1526709.1526816

[118] A. Keramatian, V. Gulisano, M. Papatriantafilou, and P. Tsigas, "PARMA-CC: Parallel multiphase approximate cluster combining," in *Proceedings of the 21st International Conference on Distributed Computing and Networking*, ser. ICDCN 2020. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3369740.3369785

[119] D. Meagher, "Geometric modeling using octree encoding," *Comput. Graph. Image Process.*, vol. 19, no. 1, p. 85, 1982. [Online]. Available: https://doi.org/10.1016/0146-664X(82)90128-9

[120] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *1984 SIGMOD Int. Conf. on Management of Data*. ACM Press, 1984, pp. 47–57. [Online]. Available: https://doi.org/10.1145/602259.602266

[121] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces," in *VLDB'97, 23rd Int. Conf. on Very Large Data Bases*. M. Kaufmann, 1997, pp. 426–435. [Online]. Available: http://www.vldb.org/conf/1997/P426.PDF

[122] R. Krauthgamer and J. R. Lee, "Navigating nets: simple algorithms for proximity search," in *Proc. of the Fifteenth Annual ACM-SIAM Symp. on Discrete Algorithms, SODA 2004*. SIAM, 2004, pp. 798–807. [Online]. Available: http://dl.acm.org/citation.cfm?id=982792.982913

[123] G. Andrade, G. S. Ramos, D. Madeira, R. S. Oliveira, R. Ferreira, and L. Rocha, "G-DBSCAN: A GPU accelerated algorithm for density-based clustering," in *Int. Conf. on Computational Science, ICCS 2013*, ser. Procedia Computer Science, vol. 18. Elsevier, 2013, pp. 369–378. [Online]. Available: https://doi.org/10.1016/j.procs.2013.05.200

[124] D. Arlia and M. Coppola, "Experiments in parallel clustering with DBSCAN," in *7th Int. Euro-Par Conf.*, ser. LNCS, vol. 2150. Springer, 2001, pp. 326–331. [Online]. Available: https://doi.org/10.1007/3-540-44681-8_46

[125] D. Müllner, "Modern hierarchical, agglomerative clustering algorithms," *CoRR*, vol. abs/1109.2378, 2011. [Online]. Available: http://arxiv.org/abs/1109.2378

[126] S. M. Savaresi, D. L. Boley, S. Bittanti, and G. Gazzaniga, "Cluster selection in divisive clustering algorithms," in *Proceedings of the Second SIAM International Conference on Data Mining, Arlington, VA, USA, April 11-13, 2002*. SIAM, 2002, pp. 299–314. [Online]. Available: https://doi.org/10.1137/1.9781611972726.18

[127] I. Walulya, D. Palyvos-Giannas, Y. Nikolakopoulos, V. Gulisano, M. Papatriantafilou, and P. Tsigas, "Viper: A module for communication-layer determinism and scaling in low-latency stream processing," *Future Gener. Comput. Syst.*, vol. 88, pp. 297–308, 2018. [Online]. Available: https://doi.org/10.1016/j.future.2018.05.067

[128] A. Keramatian, V. Gulisano, M. Papatriantafilou, and P. Tsigas, "PARMA-CC: parallel multiphase approximate cluster combining," in *ICDCN 2020: 21st Int. Conf. on Distributed Comp. and Networking*. ACM, 2020, pp. 20:1–20:10. [Online]. Available: https://doi.org/10.1145/3369740.3369785

[129] A. Beygelzimer, S. Kakade, and J. Langford, "Cover trees for nearest neighbor," in *23rd Conf. on Machine Learning*, ser. ICML '06. ACM, 2006, p. 97–104. [Online]. Available: https://doi.org/10.1145/1143844.1143857

[130] R. Weber, H. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *VLDB'98, 24rd Int. Conf. on Very Large Data Bases*. M. Kaufmann, 1998, pp. 194–205. [Online]. Available: http://www.vldb.org/conf/1998/p194.pdf

[131] A. Keramatian, "," https://github.com/amir-keramatian/IP.LSH.DBSCAN, 2022.

[132] H. Song and J. Lee, "RP-DBSCAN: A superfast parallel DBSCAN algorithm based on random partitioning," in *2018 SIGMOD Int. Conf. on Management of Data*. ACM, 2018, pp. 1173–1187. [Online]. Available: https://doi.org/10.1145/3183713.3196887

[133] Y. Zheng, X. Xie, and W. Ma, "Geolife: A collaborative social networking service among user, location and trajectory," *IEEE Data Eng. Bull.*, vol. 33, no. 2, pp. 32–39, 2010. [Online]. Available: http://sites.computer.org/debull/A10june/geolife.pdf

[134] A. Starczewski, P. Goetzen, and M. J. Er, "A new method for automatic determining DBSCAN parameters," *J. Artif. Intell. Soft Comput. Res.*, vol. 10, no. 3, pp. 209–221, 2020. [Online]. Available: https://doi.org/10.2478/jaiscr-2020-0014

[135] Y. Chen, S. Tang, N. Bouguila, C. Wang, J. Du, and H. Li, "A fast clustering algorithm based on pruning unnecessary distance

computations in DBSCAN for high-dimensional data," *Pattern Recognit.*, vol. 83, pp. 375–387, 2018. [Online]. Available: https://doi.org/10.1016/j.patcog.2018.05.030