Data Stream Affinity Propagation for Clustering Indoor Space Localization Data

by

Nasrin Eshraghi Ivari

Master of Computer Science, Universiti Putra Malaysia, 2013

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science in Engineering

In the Graduate Academic Unit of Geodesy and Geomatics Engineering

Supervisor:	Monica Wachowicz, Ph.D., Geodesy & Geomatics Engineering
Examining Board:	Ian Church, Ph.D., Geodesy & Geomatics Engineering
	Saqib Hakak, Ph.D., Department of Computer Science

This thesis is accepted by the
Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

May, 2021

Abstract

In the age of Internet of Things, the ability to find spatio-temporal patterns of people and devices moving in indoor spaces has become crucial for developing new applications. In particular, clustering indoor localization data streams has gained popularity in recent years due to their potential of generating relevant information for planning building automation, evaluating energy efficiency scenarios, and simulating emergency protocols. In this thesis, a data stream Affinity Propagation (DSAP) clustering algorithm is proposed for analyzing indoor localization data generated from e-counters and WiFi localization systems. The data sets are a sequence of potentially infinite and non-stationary data streams, arriving continuously where random access to the data is not feasible and storing all the arriving data is impractical. The DSAP algorithm is implemented based on a two-phase approach (i.e., online and offline clustering phases) using the landmark time window model. The proposed DSAP is non-parametric in the sense of not requiring any prior knowledge about the number of clusters and their respective labels. The validation and performance of the DSAP algorithm are evaluated using real-world data streams from two experiments aimed at finding stair usage patterns and occupancy behaviour in indoor spaces.

Dedication

To all the women of my land who dream of freedom

and

To my Mom and Dad for all their support

Acknowledgements

This research was supported by the NSERC/Cisco Industrial Research Chair [Grant IRCPJ 488403-14]. We would also like to thank the Flinders University for providing us with the e-counter data.

I would like to express my sincere gratitude toward my supervisor, Dr. Monica Wachowicz, for her guidance, encouragement and advice she has provided throughout my time as her student. I have been extremely lucky to have a supervisor like her.

I am grateful to my lab-mates, friends and colleagues in People in Motion Lab for many interesting discussions and sharing ideas.

I would like to express my sincere thanks to Dr. Swadesh Patra for all his support, encouragement, companionship, and dedicated time to teach me how to become a critical thinker.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Clustering is one of the most important unsupervised learning methods, which partitions a given set of data points into subsets called clusters, such that data points in the same cluster are similar and data points in different clusters are dissimilar. Clustering data streams continuously produces and maintains the clustering structure from the data stream. A data stream is a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) series of data points coming at a very high speed.

The challenges in clustering data streams are two-fold. The first one is processing the fast incoming data streams. Because of their infinite nature and rate, it is impossible to store and analyze the data streams based on an offline mode. From its inception, data streaming faces large-scale issues, and new clustering algorithms are required. The second challenge is related to the variations in the underlying data distribution due to the evolution of the clusters over time. Time window models including sliding, damped, landmark, and pyramidal, are needed to develop stream clustering algorithms (Nguyen, Woon, & Ng, 2015). These models aim to handle the evolution of the distribution of the data stream over time. Applying a time window makes it possible to analyze and store a stream within a specific time frame and

discard the previous historical data (Mansalis et al., 2018).

Previous research has proven that partitioning-based clustering models are suited for problems where the optimal number of micro-clusters is a-priori unknown. The data processing is optimized by representative a cluster-centroid for each micro-cluster. Cluster-centroids of micro-clusters have been previously explored for improving indoor fingerprinting using Wi-Fi RSS data (Hu, Shang, Gu, & Han, 2015; Subedi, Gang, Ko, Hwang, & Pyun, 2019). They have also been successfully applied for improving routing schemes for multi-level heterogeneous Wireless Sensor Networks, reducing energy consumption (Wang, Gao, Wang, Sangaiah, & Lim, 2019).

The affinity propagation algorithm has been chosen for analyzing indoor localization data in this research work. Affinity propagation (AP) clustering algorithm is a partitioning-based message passing algorithm that treats every data point as a potential centroid, and each point is given equal importance (Delbert, 2009). Also, the AP algorithm does not require the number of clusters as an input which makes any continuously operating automated operation more robust, especially if it is used on a cloud. It has been successfully applied in many scientific research and for different data sets such as intrusion detection, energy estimation, gene expression, psychology, business, physical science, and social science.

However, affinity propagation also suffers from high memory complexity, and hence its ability to handle large data sets diminishes rapidly with size and cannot handle massive data sets. However, for clustering data streams which are small data depending on the data rate and the time interval of the time window used for the clustering, a streaming AP algorithm becomes a viable solution for clustering IoT data streams such as the indoor localization data streams. To the best of our knowledge, no research work on indoor localization data streams using a streaming AP algorithm has been previously published.

This thesis proposes a novel Data Stream Affinity Propagation (DSAP) algo-

rithm using the landmark time window model for clustering indoor localization data streams obtained from two experiments deployed in indoor spaces. The DSAP model is capable of supporting the online-offline phases. In the online phase, micro-clusters are constantly computed using a landmark time window model to handle the most recent data in the stream and to continuously follow the changing data distribution. The offline phase is then performed, and the micro-clusters themselves are clustered to provide the overall clustering results. The entire online and offline phases that deliver the final clusters are performed without any user intervention. The experimental results validate the robustness of the DSAP model in handling dynamically evolving data streams based on intrinsic validation indices and performance evaluation metrics.

1.1 Research Objectives

The overall research goal is to improve current streaming AP models to generate micro and macro clusters from indoor localization data. The objectives can be described as follows.

- Develop a new data stream AP model (DSAP) for uncovering evolutionary patterns based on an online micro-cluster phase and offline macro-cluster phase.
- Apply the landmark time window model to continuously accumulate sufficient data points for computing the micro-clusters, avoiding performance issues.
- Evaluate the clustering results using performance and validation metrics that can be used to infer the advantages and limitations of the proposed DSAP model.
- Demonstrate the potential of DSAP for analyzing localization data streams in order to understand staircase usage patterns as well as occupant behavior in

indoor spaces.

1.2 Scientific Contributions

The main scientific contributions of this research work are summarized as follow:

- A new data stream AP clustering (DSAP) algorithm is proposed for uncovering evolutionary patterns using the landmark time window model. Previous research work was focused on using the sliding time window model.
- To the best of our knowledge, the streaming AP clustering algorithms have never been used for clustering indoor localization data streams before.
- We demonstrate the potential of applying streaming cluster analysis to find new insights into stair usage and occupant behaviour in indoor spaces.

1.3 Organization of this Thesis

This thesis is organized into five chapters as described as follows:

Chapter 2 provides an overview of clustering methods by reviewing the various concepts, methods, and underlying steps, with a particular focus on the AP method. Data stream clustering algorithms are also introduced with the time window models that are used for computing stream clusters.

Chapter 3 summarizes the related work in streaming AP clustering.

Chapter 4 describes the research methodology adopted for developing, implementing, and evaluating the proposed Data Stream Affinity Propagation (DSAP)algorithm using the landmark time window model.

Chapter 5 describes the design and implementation of the DSAP algorithm. Two experiments are presented that are related to applying the proposed DSAP model to find stair usage patterns and occupant behaviour in indoor spaces.

Chapter 6 provides an in-depth discussion of the main clustering results from applying DSAP using the data streams generated from both experiments.

Finally, Chapter 7 concludes the work and outlines future research work.

Chapter 2

Background

This Chapter presents an overview of clustering methods by reviewing the various concepts, methods, and underlying steps, with a particular focus on the affinity propagation (AP) method. Data stream clustering algorithms are also introduced with the time window models that are used for computing stream clusters.

2.1 Cluster Analysis

2.1.1 Overview

Clustering refers to partitioning a set of data points (also referred to as observations or tuples) into groups according to a proximity measure. Distance functions are predominantly used to determine a proximity measure of dissimilarity/similarity between clusters. This proximity measure directly affects the formation of the resulting clusters. Almost all clustering algorithms are explicitly or implicitly connected to some definition of distance function (Zumel, Mount, & Porzak, 2014). The most commonly used distance functions for quantitative feature spaces are summarized in Table 2.1.

Figure 2.1 illustrates the results of a clustering process for a set of points in a

Table 2.1: Main distance functions used for computing proximity measures

Distance	Function	Description
Euclidean distance	$\sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$	The root square differences between co-ordinates of a pair of data points, most applied way of representing distance in clustering. Tend to form hyperspherical clusters.
Manhattan distance	$\sum_{i=1}^{n}\|q_i - p_i\|$	The sum of the distances in each dimension in a n-dimensional space
Minkowski distance	$(\sum_{i=1}^{n}\|q_i - p_i\|^n)^{\frac{1}{n}}$	The generalization of the Euclidean distance and the Manhattan distance
Cosine distance	$1 - cos\alpha = \frac{q_i^T - p_i}{\|q_i\|\|p_i\|}$	A metric for measuring distance when the magnitude of the vectors does not matter, commonly used in text data
Mahalanobis distance	$\sqrt{(q_i - p_i)^T S^{-1}(q_i - p_i)}$	The distance between a data point and the mean, requiring high computational complexity

2D space, where different data points were partitioned into three clusters using the Euclidean distance. Each cluster is represented by its centroid or exemplar (red data points). In general, the intra-cluster data points are similar due to their proximity in the feature space, meanwhile the inter-cluster data points are dissimilar.

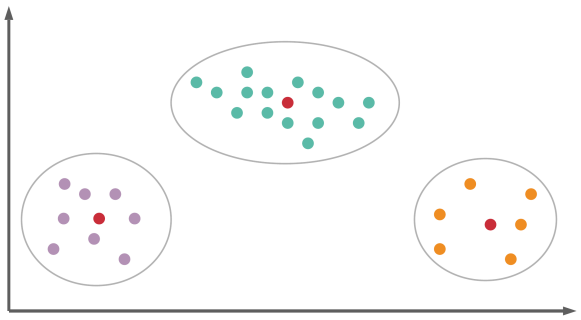


Figure 2.1: Clustering in 2D feature space

Once a proximity measure is chosen, the construction of a clustering criterion function makes the partition of clusters an optimization problem, which is well defined mathematically, and has rich applications in marketing, biomedical and geospatial fields (Jain, Murty, & Flynn, 1999).

2.1.2 Clustering Methods

Clustering is ubiquitous, and a wealth of clustering methods has been developed to solve different problems in specific fields. However, there is no clustering method that can be universally used to solve all problems. Some problem examples can be described as follows:

- *Group Discovery*: Clusters provide useful information for supporting a preprocessing step of discriminant analysis.
- *Pattern Recognition*: Clusters can be used for supporting template matching by determining similarities between two entities of the same type (e.g., points, curves, or shapes), finding patterns and regularities in data.
- *Data Compression*: Clusters provide a summary of a data set by reducing the number of data points needed to represent a data set. This is achieved by using cluster centroids rather than the actual data points.
- *Outlier detection*: Clustering has been successfully applied to find outliers in data. An outlier can be assigned to a data point very far from its cluster centroid or a cluster that deviates from the prominent clusters.
- *Feature Reduction*: Clusters can be used for dimension reduction when the number of features in a data set is larger than the number of data points.

Therefore, selecting a clustering method is an important step, as certain methods might be best-suited for a particular type of data sets in order to reflect the

true nature of the data (Berkhin, 2006; Han, Pei, & Kamber, 2011). Traditionally, clustering methods can be distinguished into five main categories described as follows:

- *Partitioning methods:* In these methods data points are split into k number of clusters. Each data point can only belong to one cluster, and each cluster must contain at least one data point. The algorithms usually require a distance threshold to generate new clusters, specially those with spherical shapes. K-means is the most recognized clustering algorithm in this category due to its robustness in finding clusters that have not been explicitly labeled in the data.

- *Hierarchical methods:* The aim is to build a tree-like nested structure from a set of data points (Swarndeep Saket & Pandya, 2016). Initially each data point is considered as an individual cluster, and at each iteration, a proximity matrix is computed to merge similar clusters until one cluster or k clusters are formed. These methods are widely used to analyze social network data. The BIRCH algorithm is an example of a hierarchical method where every data point is equally important for clustering purposes. Moreover, the concept of Clustering Feature (CF) is fundamental for summarizing the information that needs to be maintained about a cluster.

- *Density-based methods:* Clusters are separated from each other by contiguous regions of low density of data points. They are highly efficient to find arbitrary shaped clusters and clusters with noise. DBSCAN and OPTICS are well-known algorithms in this group.

- *Grid-based methods:* The feature space is partitioned into a finite number of cells to form a grid structure and then find the clusters from the cells. These methods are efficient in mining large multidimensional data sets. STING is one

of the highly scalable algorithms in this group with the ability to decompose a feature space into various levels of detail.

- *Model-based methods:* The main assumption is that the data points are generated by a model which can be adapted to what we know about the underlying distribution of the data. The model that we recover from the data then defines clusters and an assignment of data points to new clusters. A commonly used criterion for estimating the model parameters is maximum likelihood.

The main advantages and disadvantages of these different clustering methods are summarized in Table 2.2 (Mousavi, Bakar, & Vakilian, 2015).

Table 2.2: Comparison of clustering methods

Categories	***Advantages***	***Disadvantages***
Partitioning Methods	* Easy to implement * Produce tighter clusters	* Best suited for finding spherical shaped clusters
Hierarchical Methods	* Easy to handle any type of distance function	* High computational complexity
Grid-based Methods	* Fast processing time * Effective with handling noisy data	* Limited to the Euclidean distance function
Density-Based Methods	* Fast to compute * Easy to interpret clusters	* Higher value for convex clusters compared with density-based clusters
Model-Based Methods	* Offer more flexibility * Effective in handling noisy data	* Highly dependent on the hypothesized model

2.1.3 Affinity Propagation Clustering

Affinity propagation (AP) is a partitioning-based clustering algorithm that was first introduced by Frey and Dueck in 2006 (B. J. Frey & Dueck, 2006). AP is based on the concept of "message passing" between data points that plays an important role in finding the best candidate data point that becomes a cluster centroid (Frey, 2007; Jiang, Bi, Xia, Xue, & Qian, 2019).

Unlike clustering algorithms such as k-means, affinity propagation does not require the number of clusters to be determined or estimated before running the algorithm. It considers all data points as a potential centroids until an optimal set of clusters and their respective centroids are found through an iterative process.

The AP algorithm supports similarities that are not symmetric, and it is not dependent on initialization found in other clustering algorithms (Refianti, Mutiara, & Gunawan, 2017). In order to apply AP for any data set two conditions should be met: no missing or null data in the inputs and the feature space should contain only numeric, not categorical data. However, it has a significant speed problem, particularly for clustering large volume of data. Affinity Propagation needs $O(N^2T)$ time to update a message where N and T represent the number of data points and the number of iterations (Frey, 2007).

The AP algorithm is based on the computation of four matrices: similarity matrix, responsibility matrix, availability matrix, and criterion matrix. They are described as follows:

- *Similarity Matrix:* Every cell value in the similarity matrix $s(i, j)$ corresponds to an element k representing how similar/dissimilar two data points are in a feature space. This element k is defined as the negative of the Euclidean distance between the two data points.

$$s(i, j) = -\|x_i - x_j\|^2 \tag{2.1}$$

The greater the distance between any two data points, smaller is the k between them. The k values of the off-diagonal cells will dictate the number of clusters formed, in such a way that the smaller the index value (value ≤ 0), fewer number of clusters is obtained.

- *Responsibility Matrix:* The matrix $r(i,k)$ quantifies how well-suited element k is, to be a cluster centroid for element i, taking into account the nearest contender k' to be a cluster centroid for i.

  We initialize the matrix R with zeros, and compute the next values using the equation:

$$r(i,k) \leftarrow s(i,k) - \max_{k' s.t. k' \neq k} \{a(i,k') + s(i,k')\} \tag{2.2}$$

  The goal is to quantify how similar is i to k, compared to some k', taking into account the availability of k'.

- *Availability matrix:* The matrix $a(i,k)$ quantifies how appropriate is it for i to choose k as its cluster centroid, taking into account the support from other elements that k should a centroid. Availability is self-responsibility of k plus the positive responsibilities of k towards elements other than i as shown in the following equation:

$$a(k,k) \leftarrow \sum_{i' \neq k} \max(0, r(i',k)) \tag{2.3}$$

  The R and A matrices are iteratively updated. This procedure may be terminated after a fixed number of iterations, after changes in the values obtained fall below a threshold, or after the values stay constant for some number of iterations.

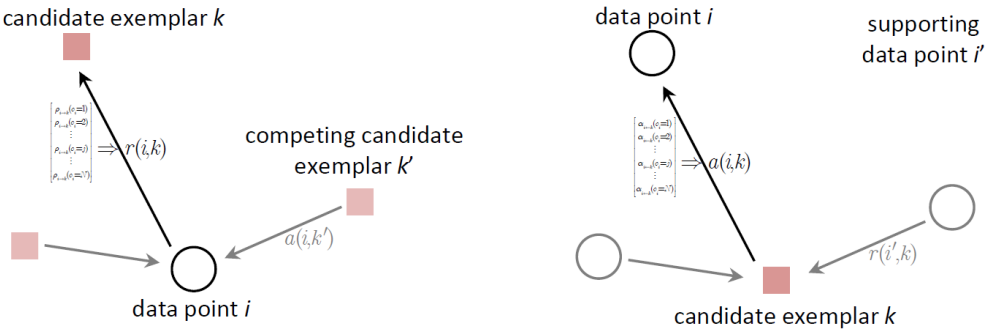


Figure 2.2: Message passing based on responsibility and availability matrices

Figure 2.2 illustrates how the AP algorithm sends responsibility and availability messages to data points (Delbert, 2009). Responsibility $r(i,k)$ is the message from data point i to candidate centroid k and answer relies on determining if point k is well-suited to be a centroid for data point i. Availability $a(i,k)$ sends a message from centroid k to data point i to find how well it would be for point i to choose point k as a centroid.

- *Criterion matrix:* This matrix is calculated after the iteration process is terminated.The $c(i,k)$ matrix is the sum of R and A. An element i will be assigned to an exemplar k which is not only highly responsible but also highly available to i as shown in

$$c(i,k) \leftarrow r(i,k) + a(i,k) \tag{2.4}$$

The element with the highest criterion value in each row would be designated to be a cluster centroid. Elements that have the same centroid will be in the same cluster.

The AP clustering algorithm used in this thesis is described in Algorithm 1. The Python code used in this work is available at https://scikit-learn.org/ open source repository. Four hyperparameters are used for tuning the clustering process using AP. They are described as follows:

- Damping Factor is the extent to which the current value is maintained relative to incoming values (weighted 1 - damping). This is used to avoid numerical oscillations when updating messages. The range of this hyperparameter is between 0.5 to 1.

- Maximum number of iterations.

- Convergence is the number of iterations with no change in the number of estimated clusters that stops the convergence.

- Preference values indicates how likely a point is to be a centroid itself. If the preferences are not passed as arguments, they will be set to the median of the input similarities.

Algorithm 1: Affinity Propagation Algorithm

Input: Data $d = (d_1, d_2, ..., d_n)$
Hyper-parameters: Preference, Damping, $conv_{it}$, max_{iter}
Output: Cluster centroids $C_1, ..., C_k$, branch points
`AP(`d`)` **Initialization:**
Similarity Matrix: S $\forall$ i, k: s(i, k) = 0
Availability Matrix: A $\forall$ i, k: a(i, k) = 0
Responsibility Matrix: R $\forall$ i, k: r(i, k) = 0
Criterion Matrix: E $\forall$ i, k: e(i, k) = 0
Compute S: $\forall i, k : s(i, k) \leftarrow -\|V_i - V_k\|^2 where V_i = (M_i, S_i)$
while $r(i, k)$ & $a(i, k) \neq convergence$ **do**
  Compute R:
  $r(i, k) \leftarrow s(i, k) - \max_{k' s.t. k' \neq k} \{a(i, k') + s(i, k')\ \}$
  Compute A:
  $a(i, k) \leftarrow \min\{0, r(k, k) + \sum_{i' s.t. i' \notin \{i,k\}} \max\{0, r(i', k)\}$
  no-diagonal A:
  $a(i, k) \leftarrow \sum_{i' \neq k} \max(0, r(i', k))$
  cluster assignment: $E = (E_1, ..E_n)\ E = argmax[a(i, k) + r(i, k)]$
Results C = $(C_1, ..., C_k)$

2.2 Streaming Cluster Analysis

2.2.1 Overview

In almost all clustering methods described in the previous section, the number of data points is considered fixed, and each data point is used multiple times for computing proximity measures and matrices in order to be assigned to a particular cluster. These methods are computationally high and require large data storage.

However, with the advent of Internet of Things, and in particular indoor localization systems, a wide range of sensor technologies (e.g., e-counters, environmental, and motion sensors) and networking communication technologies (e.g., infrared, RFID, sensors, WiFi, and Bluetooth) are being used for sensing and transporting large volumes of data. The data are harvested as data streams that are a sequence of digitally encoded coherent signals (packets of data or data packets). These data packets arrive according to different data rates (i.e. bandwidth), which are often expressed in bytes per second (B/s).

From a clustering perspective, a data stream S is sequence of infinitive, ordered, and fast-changing stream data points $d_1, d_2, ..., d_3 \xrightarrow{S} d_i$ (Han et al., 2011). Therefore, clustering methods need to form clusters from a continuous flow of stream data points. The traditional set-up where an entire static data set is available for clustering can not be applied to data streams which continuously arrive at a rapid rate. The main issues can be summarized as follows (Toshniwal, 2013):

- Previous data used in clustering analysis are static, but data streams are unbounded and non-stationary data.

- Streaming cluster analysis requires a process capable of partitioning the data streams continuously while taking into account restrictions of memory and time. Traditional clustering methods partition an entire data set at once, and memory requirements do not change over time.

- Traditional clusters are found from accumulated data points, but streaming clusters evolve over time depending on the time window model being used to accumulate the data points.

It is important to select a time-interval for a time window model based on the trade-off between the data rate and latency, as that plays an important role in controlling which data points in the stream are going to be processed at the current time. Four main types of time window models have been proposed in the literature (Nguyen et al., 2015; Mansalis et al., 2018). They can be described as follows:

- *Damped Time Window:* It is also referred to as time-fading model. Each data point has a different weight based on its arrival time so new data points have a higher weight than the old ones. This time window model reduces the impact of old data. It is mostly applied using density-based clustering methods. The function $f(\delta t) = \mu_{\delta t}$ $(0 < \mu < 1)$ is usually used for this model, where δt is the age of a data point which is equal to time differentiation between current time and arrival time. A fading parameter μ is in the range between 0 to 1.

- *Landmark Time Window:* It is also known as the hopping model, and is used when clustering starts from a starting point call landmark to the current time. When a new window comes, all data points from the previous landmark time window are removed. All data points are equally important.

- *Sliding Time Window:* The most important data points are the most recent ones and old data points will be removed. The structure of the sliding time window is based on the FIFO (First-In-First-Out) principle, in which a data point from the current time is used for clustering until a specific time is passed. The window has a size of w in the current time t, will slide (s, sliding(w)) where s[t-w+1, t]. The window size can be set based on resources, and computational

process (Silva et al., 2013). This model is efficient when most recent data points are relevant for the cluster analysis.

- *Pyramidal Time Window:* It is also known as tilted time window and focuses on recent data points without discarding old data points. It applies various time granularity levels based on how recent a data point is (Aggarwal, Philip, Han, & Wang, 2003; Nguyen et al., 2015). It stores almost all data points belonging to a specific time window, and after computes a balance between storage requirements and accuracy. The old data tuples are aggregated.

Figure 2.3 illustrates how these time window models have been conceptualized. Time window models also handle concept-drift by changing the trends and data distribution in clustering, avoiding forming less accurate clusters as time passes. Finding patterns without storing all the stream data points is one of the main tasks of streaming cluster analysis.

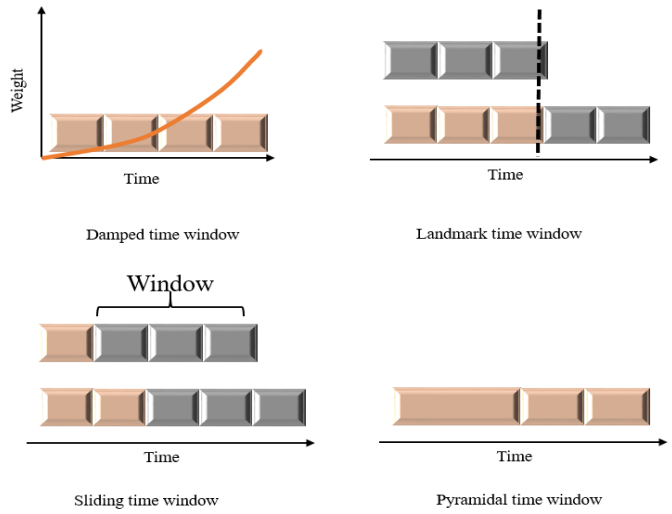


Figure 2.3: Current time window models used with stream data points (Carnein & Trautmann, 2019)

Processing of data points using time window models can take two opposite views of the streaming problem as pointed out by Mansalis et al. (2018). The first view is to use a two-phase scheme which consists of an online component that processes

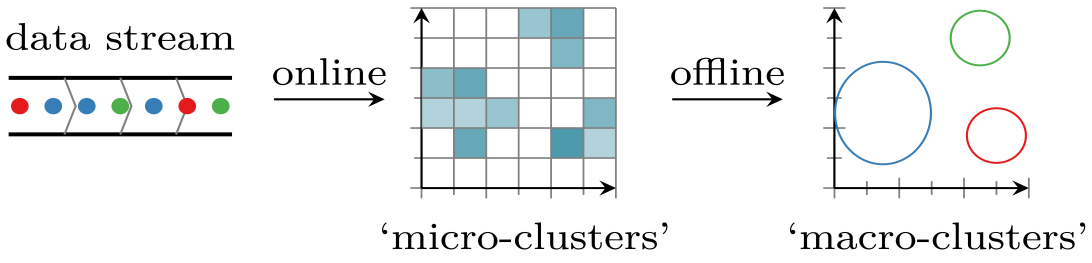


Figure 2.4: Overview of the two-phase processing strategy (Carnein & Trautmann, 2019)

stream data points and produces summary data (e.g., micro-clusters, core-micro-clusters, temporal CF, grids, and coreset tree); and an offline component that uses the aggregated summary data to form the final clusters. In the literature of data stream clustering methods, a large number of algorithms use a two-phase scheme as shown in Figure 2.4 .

An alternative view is to generate final clusters without the need of an offline phase. A single-pass model is maintained over a stream, and clusters are updated as new data points arrive from the stream. One of the main disadvantages of this strategy is that the evolution of the clusters can not be analyzed at different time intervals; only a final snapshot is produced.

In this thesis, the online/offline strategy was adopted, and final clusters are referred to as macro-clusters; meanwhile the the centroids of micro-clusters were used as summary data. The online component requires a process for saving the summary data in a fast manner, and is not requested to support real-time processing, but rather a very low response delay. The offline phase uses the aggregated summary data to compute the macro-clusters. It is not time-related and can happen any time by user request.

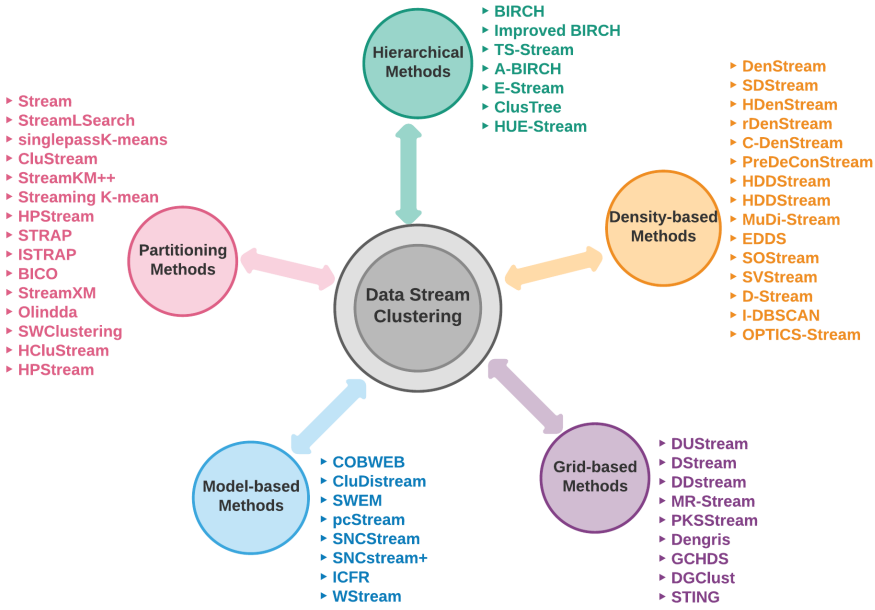


Figure 2.5: Data stream clustering methods and algorithms

2.2.2 Stream Clustering Methods

Data stream clustering methods can be categorized into five leading groups according to the nature of their underlying clustering methods as previously described in Section 2.1.2. Figure 2.5 provides an overview of the main methods developed within these categories. Table 2.3 provides a comprehensive overview of the data stream clustering methods.

In this research work, the DSAP algorithm is proposed as a simplification of the previous streaming AP algorithms. This is further discussed in the next chapter.

Table 2.3: Overview of data stream clustering methods (Mansalis et al., 2018)

Algorithms	Year	Approach	Processing	Time Window
Partitioning-based				
Stream	2000	k-median	Single-pass	Landmark
StreamLSearch	2002	k-median	Single-pass	Landmark
CluStream	2003	k-means	Online-Offline	Pyramidal
HCluStream	2006	k-means	Online-Offline	Pyramidal
Olindda	2007	k-means	Online-Offline	Landmark
SWClustering	2008	k-means	Online-Offline	Sliding
StreamKM++	2012	k-means++	Online-Offline	Landmark
BICO	2013	k-means	Online-Offline	Landmark
StreamXM	2015	x-means	Merge-reduce	Landmark
Hierarchical-based				
BIRCH	1996	CF-tree	Single-pass	Landmark
TS-Stream	2015	Decision Tree	Single-pass	Sliding
Density-based				
DenStream	2006	DBSCAN	Online-Offline	Damped
SDStream	2009	DBSCAN	Online-Offline	Sliding
HDenStream	2009	DBSCAN	Online-Offline	Damped
FlockStream	2009	Swarms	Single-pass	Damped
rDenStream	2009	DBSCAN	Online-Offline	Damped
C-DenStream	2009	C-DBSCAN	Online-Offline	Damped
PreDeConStream	2012	DBSCAN	Online-Offline	Damped
HDDStream	2012	DBSCAN	Online-Offline	Damped
MuDi-Stream	2016	DBSCAN	Online-Offline	Damped
EDDS	2017	DBSCAN	Online-Offline	Damped
Grid-based				
DUstream	2005	Dense-unit	Online-Offline	Landmark
D-stream	2007	Dense-region	Online-Offline	Damped
DDStream	2008	DCQ-means	Online-Offline	Damped
MR-Stream	2009	Dense-region	Online-Offline	Damped
PKS-Stream	2011	Dense-region	Online-Offline	Damped
DENGRIS	2012	Dense-region	Single-pass	Sliding
Model-based				
COBWEB	1987	Tree-based	Online-Offline	Landmark
SWEM	2009	EM Algorithm	Online-Offline	Sliding
pcStream	2015	SIMCA	Single-pass	Damped
SNCStream	2015	Network	Online	Damped
SNCStream+	2016	Network	Online	Damped

Chapter 3

Literature Review

Indoor localization technologies have been a key enabler for advancing the usefulness of the Internet of Things in biomedical and health care applications. Many localization technologies have been explored for generating location data of people in indoor spaces, including WiFi, BLE beacons, RFID tags, visible light wave, and ultra-wideband (Namiot, 2015; Jeon, She, Soonsawad, & Ng, 2018).

In particular, infra-red, optical (e.g., RGB videos or flat images), break beam, thermal, and ultrasonic sensors have been used for counting people in indoor spaces (Mautz, 2012). One simple way of counting people is using ultra-wideband radar sensors mounted in e-counter devices that can be used for counting multiple people passing through a passage or a wide door. Two sensors equipped with antennas which have narrow beam width are used to form two invisible electronic layers in the path. These layers are used for sensing the presence of a person and recognizing the direction of the movement. E-counters are usually deployed at stairs and the entrances of indoor spaces.

With the increasing proliferation of wireless LAN, WiFi signal strength measurements from access points have been used for localization. RSSI (Received Signal Strength Indicator) is a measurement of how well a mobile device can receive a signal

from an access point or router. The IEEE 802.11 standard specifies that RSSI can be on a scale of 0 to up to 255 and that each chipset manufacturer can define their own "RSSI Max" value. Cisco, for example, uses a 0-100 scale, while Atheros uses 0-60. Vendors and chipset makers provide their own accuracy, granularity, and range for the actual power (measured as milliwatts or decibels) and their range of RSSI values (from 0 to RSSI maximum).

Previous research work applied the AP partitioning-based method for improving indoor fingerprinting using WiFi RSS data (Hu et al., 2015; Subedi et al., 2019). They have also been successful applied for improving routing schemes for multi-level heterogeneous Wireless Sensor Networks, saving energy consumption (Wang et al., 2019). In this thesis, we are actually interested in clustering WiFi location data. The underlying assumption is that for each location of a mobile device in an indoor space, the WiFi RSS signals of these mobile devices are different. By relying on the variation of these signals in a distinct indoor space, the current location of a mobile device can be obtained.

The stream data points generated from e-counters and WiFi localization systems require streaming clustering since they are a sequence of potentially infinite, non-stationary data (the probability distribution of the unknown data generation process may change over time) arriving continuously (which requires an online phase through the data) where random access to the data is not feasible and storing all the arriving data is impractical (which requires an offline phase). To the best of our knowledge, streaming AP algorithms have not yet been applied for clustering e-counters and WiFi location data generated in indoor spaces.

Overall, there are a few robust streaming algorithms based on the AP method. The clustering process requires a strategy capable of partitioning stream data points continuously while taking into account restrictions of memory and time. Table 3.1 summarizes the main characteristics of four streaming algorithms that were proposed

in previous research works.

Table 3.1: Overview of streaming AP clustering algorithms

Algorithms	**Year**	**Method**	**Processing**	**Time window model**
StrAP	2008	AP	Single-pass	Sliding
IStrAP	2012	AP	Single-pass	Sliding
ISTRAP	2018	AP	Single-pass	Sliding
APDenStream	2013	AP	Online-offline	Sliding (online) Pyramidal (offline)

Overall, these streaming AP algorithms usually suffer from a quadratic computational complexity in the number N of items. The time complexity of AP is $O(n^2 * T)$, where n is the number of samples and T is the number of iterations until convergence happens(Refianti et al., 2017). This computational performance might become an issue for computing macro-clusters using large volumes of people counting data. It is important to develop strategies to improve the performance of AP models.

StrAP Algorithm

StrAP was the first algorithm proposed to extend affinity propagation for streaming data and was inspired by the DenStream algorithm (X. Zhang, Furtlehner, & Sebag, 2008). It has a simple evolutionary workflow as illustrated in Figure 3.1 where a set of 4-tuples $(c_i, n_i, \sum i, t_i)$ is represented, where c_i ranges over the clusters, n_i is the number of data points associated to cluster c_i, $\sum i$ is the distortion of c_i ($\sum d(x, c_i)^2$), where x ranges over all data points associated to c_i, and t_i is the last timestamp when a data point was associated to c_i.

The number of clusters to be found is not pre-determined, and the clustering problem is defined in terms of energy minimization. An initial number of actual data points are selected as exemplar cluster-centroids, and the algorithm computes the cost of adding another new cluster-centroid. The penalty parameter value controls

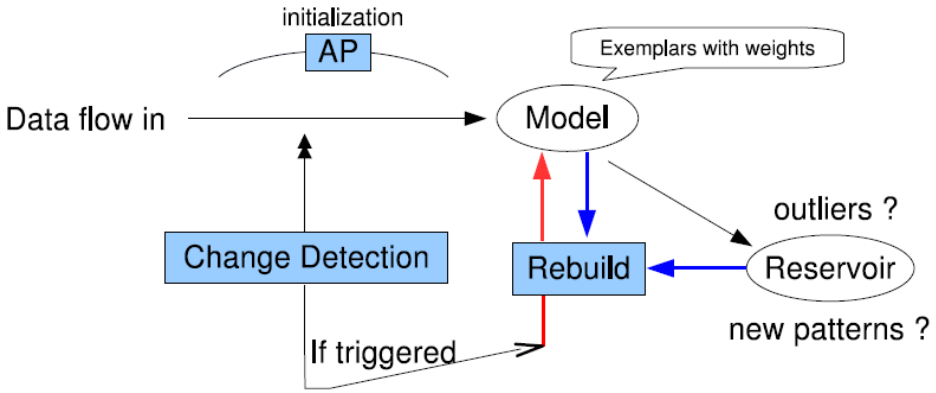


Figure 3.1: The workflow developed for the StrAP algorithm

the cost of adding another exemplar. If the current data point does not fit the model according to its penalty value, it is automatically moved to a reservoir. A statistical test enables StrAP to catch drifting exemplars that significantly deviate away from the existing clusters. StrAP involves four main steps described as below:

- A first batch of data stream data points is selected and the first cluster-centroids are obtained using AP to initialize the StrAP algorithm.

- As new stream data points arrive, each stream data point x_t is compared to the existing centroids e_i. This comparison is carried out by computing the Euclidean distance of x_t, e_i with the threshold ϵ, which is a heuristically set, and the value is equal to the average distance between stream data points and centroids in the initial step. If x_t is far from the nearest centroid, it goes to the reservoir. If not, x_t is assigned to the i^{th} cluster, and the clusters are updated accordingly. StrAP has a mechanism to forget old centroids which have not been visited for some time.

- Two restart procedures have been proposed in StrAP. First, if the number of outliers in the reservoir exceeds the reservoir's size, the restart criterion is triggered. Second, the Page-Hinkley (PH) test is used for detecting an abrupt change in the average value of the data points in clusters.

- If the restart criterion is satisfied either due to the reservoir filling up or if the PH test detects an abrupt change, the clusters are rebuilt by launching a weighted version of AP (WAP) using current centroids and the reservoir.

Previous research has explored the trade-off between the computational cost and the performance of StrAP. The distortion and purity metrics have been applied to evaluate the quality of the clustering results, using data sets such as the Intrusion Detection benchmark data (KDD99), synthetic stream data, and the streaming data from the EGEE grid system (X. Zhang et al., 2008).

Artificially generated data streams was used for testing the ability of StrAP in handling a dynamically changing data distribution (X. Zhang et al., 2008). The network connection data, KDD Cup 1999 (KDD99) Intrusion Detection was also used to evaluate the StrAP model for distinguishing between "bad" connections, called intrusions or attacks, and "good" normal connections. Each connection was labeled as one of 23 classes, the normal class and the specific kinds of attack. StrAP's clustering quality was tested with data streams containing EGEE jobs in the whole EGEE grid over 5 months. Besides the six attributes related to time-cost, each job was labeled by its final state, successfully finished (good job) or failed (bad job, including about 45 error types). The 5 million jobs include about 20 main error types (more than 1,500 occurrences). The job labels were used as an indicator of the clustering quality.

While the accuracy of StrAP was found satisfactory, the computational time was high, hampering its further use with large volumes of stream data.

IStrAP and ISTRAP Algorithms

IStrAP introduced a simpler method to eliminate outliers as compared with the existing technique used in StrAP (Li & Li, 2012). An outlier here refers to an abnormal cluster that represents a behavior deviating from the normal clusters. The

reservoir in StrAP emulates the sliding time window model, making clustering ineffective for handling outliers in the cached data. IStrAP relies on statistical analysis of the stream data points that are temporarily stored in the reservoir, removing outliers from the reservoir according to their statistical properties, and then clustering the remaining stream data points left in the reservoir. The process of removing outliers consists of four steps described as follows:

- Compute the reservoir's statistics such as maximum, minimum, sum, average, and standard deviation.
- Scan the reservoir to calculate the standard deviation of data points.
- Set the standard deviation as the sampling parameter (σ) to reflect the distribution of the data in reservoir.
- Remove outliers if they are outside of the interval [Avg-σ, Avg + σ].

The accuracy metric has been used to evaluate the quality of the clustering results and to compare the clustering results from IStrAP and StrAP. The Intrusion Detection benchmark data (KDD99), which distinguishes the attack from normal connections out of labels was used in the evaluation (Li & Li, 2012). The experimental results show that IStrAP can eliminate outliers, and it has shown a higher clustering accuracy than the StrAP algorithm. This approach works well when the data has a normal distribution.

Another attempt to improve the StrAP algorithm is towards inheriting its advantages but providing a better performance using a dynamic clustering scheme, which ensures that not only new emerging clusters can be detected, but disappeared and recurrent clusters are also timely detected. In the ISTRAP algorithm (Sui, Liu, Jung, Liu, & Li, 2018), cluster disappearance refers to an existing cluster that is not visited by any recently arrived data points. Cluster re-occurrence occurs when a previously disappeared cluster reoccurs at a later timestamp.

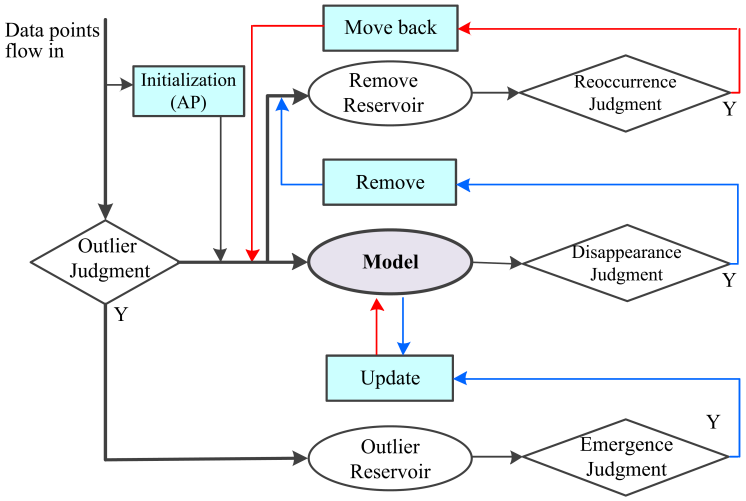


Figure 3.2: The proposed ISTRAP clustering workflow

Outdated clusters are removed and recurrent clusters are efficiently detected rather than being treated as novel clusters. To achieve this, ISTRAP defines the clusters as active and inactive. Active state means that a cluster is still valid at the current timestamp because at least one data point has been assigned to it during the current time interval. Inactive state indicates the clusters is expired since it did not gain any recent data points. Two reservoirs are proposed: an outlier reservoir for saving outliers and the remove reservoir for storing inactive clusters. The workflow of the ISTRAP algorithm is illustrated in Figure 3.2. The main steps are described as follows:

- First, initial exemplars are obtained by applying the AP algorithm on the first batch of stream data points.
- As the stream flows in, each data point goes through the outlier judgment, and the minimum distance of the data point from the exemplar is compared against a threshold. If the distance is less than the threshold, the data point is assigned to the nearest cluster; otherwise, it is sent to the remove reservoir.
- All clusters in remove reservoir are checked if they re-occur, the recurrent clusters are moved back into the model.

- The emergence criterion is triggered if new clusters need to be obtained. The model is rebuilt by the WAP algorithm using the data points stored in the outlier reservoir.

- The model checks all current clusters if they are still active or not. The inactive clusters are removed and sent to the removed reservoir.

Four parameters, δ, α, β, λ, are needed to monitor the clustering process in ISTRAP. The number of outliers is controlled by δ. The emergence threshold for detecting new clusters is defined by α, and it affects the stability and processing time of the algorithm. The maximal duration tolerance β means an active cluster is not visited by new data points. Larger β values mean less possibility that clusters are inactive. The last parameter λ represents the minimum number of data points assigned if recurrent clusters have to be sent to the model. If λ values are small, it means more recurrent clusters are detected.

To test the effectiveness of the emergence, disappearance, and recurrent detection of ISTRAP, the AS synthetic and the MNIST real-world data sets were used in the evaluation. The MNIST data set contains images of handwritten digits and their corresponding labels.The results show that ISTRAP and StrAP can detect emergence accuracately; however, the disappearance was not detected by StrAP.

ISTRAP improves the current state of the StrAP model and also the quality of the clusters, but this comes at the cost of introducing additional parameters that have increased the model's complexity and made it not fully automated, which is one of the main goals of a streaming AP algorithm. Moreover, the time complexity have increased compared to the StrAP algorithm due to the computational costs for computing two reservoirs.

APDenStream Algorithm

The APDenStream Algorithm is a data stream clustering algorithm based on

AP and density methods to improve the StrAP model's accuracy and timeliness in a noisy environment (J.-P. Zhang, Chen, Liu, & Li, 2013). This algorithm employs a two-phase clustering approach, the online and offline phases. The online phase detects data distribution changes due to the new data points, updates the micro-cluster decay density, and generates real-time results. The offline phase, which is invoked by a user who has to choose a specific time frame by applying a pyramidal time window model, gives final query results.

The clustering process of APDenStream is depicted in Figure 3.3, and it consists of the following steps:

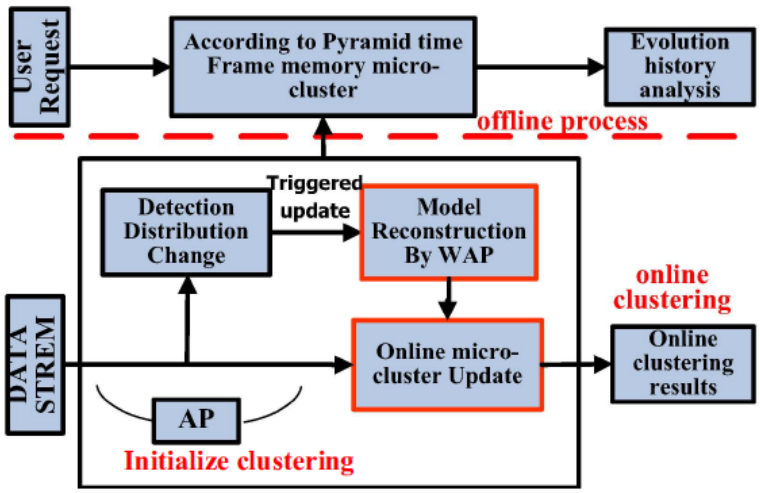


Figure 3.3: The APDenStream clustering process

- AP algorithm is applied to the first $InitN$ points P to initialize the online process. The first group of micro-clusters mc are obtained by scanning P. If the micro-cluster density of the micro-cluster mc is greater than the threshold D, it is labeled as a dense micro-cluster, and the average radius of all dense micro-clusters is being chosen as an initial radius ϵ.

- As a new data point comes into the model, and it is placed in its nearest dense or sparse micro-cluster. If it is not assigned to any existing micro-cluster, it goes to the reservoir. As time passes, the number of micro-clusters increases.

To keep the number of clusters in check and to avoid memory problems, a dynamic online maintenance strategy is used. First, search for dense micro-clusters in order to determine whether to remove them. Second, search for sparse micro-clusters by checking the minimum weight limit of sparse micro-clusters.

- Whenever the reservoir is full, the model is rebuilt, and new clusters are added to the model by applying the WAP algorithm.

- The last step is offline processing which can master a depth-first traversal to find dense micro-clusters using a snapshot at time t_c and time $t_c h$ can be found in the pyramidal time frames according to the current time t_c and user-specific time range h.

For the evaluation of this algorithm, two data sets were used the KDDCUP'99 dataset, and a synthetic Gaussian data. The results show that the APDenStream is more accurate than the StrAP algorithm due to the proposed summary data structure and its online elimination strategy, which maintains the potential micro-clusters in the stream in the time of removal of outlier noise point. Furthermore, APDenStream can remove outliers without having to upgrade potential noise points timely, while in StrAP historical expired micro-clusters are taken into account.

Conclusions

The differences among the algorithms described above are mainly reflected in the update strategy applied in the processing the data streams (i.e. single-pass or online/offline). Our focus is on designing a novel update strategy for the clustering process, allowing a streaming AP algorithm to handle evolving clusters in a simpler way and thus become more practical.

Our proposed DSAP algorithm is actually a simplified approach in compari-

son to IStrAP, ISTRAP, and APDenStream. Our research premise is that indoor localization data streams have a unique time-evolving cluster structure that does not require update strategies such as the WAP algorithm, outlier repository, decay function, and reocurrence judgement rules to handle outliers.

Mainly because the constrained indoor spaces where the data streams are collected make the occurrence of outliers extremely low. This is usually not the case in social media networks such as Twitter, where timely topics appear quickly (cluster emergence) while older topics lose relevance over time (cluster disappearance). To the best of our knowledge, StrAP, IsrAP, ISTRAP, and APDenStream have not yet been applied for indoor localization data streams. Unfortunately, the programming codes of these algorithms were not provided by their authors to us, hampering their evaluation in terms of finding streaming clusters from indoor localization data.

Moreover, DSAP aims to handle the flow of stream data points by proposing the landmark time window model for the online phase of the clustering process. Out of the previously proposed algorithms, none of them have explored the landmark time window before. Mainly because StrAP, IsrAP, ISTRAP, and APDenStream take the opportunity to use a repository to emulate the sliding time window model in the single-pass/online phase.

The landmark time window model separates the stream data points based on a landmark time interval (e.g. every minute) or by a landmark event (e.g. people are not detected in an indoor space), when after a landmark is reached, new data points start to be captured in a separate window. This strategy is particularly advantageous for defining temporal relationships among arriving data points since it creates a meaningful sequence of time order snapshots. Once a set of new data points has been gathered for one active time window, the evolution of cluster relationships over time can be generated.

Furthermore, the DSAP algorithm allows the user to decide to keep a number of

landmark time window models active in order to avoid removing historical clusters, and ensure that historical discovered clusters are equally important in the clustering process. Basically the number of active time windows dictates how much data will be used continuously for detecting new clusters. To the best of our knowledge, this type of strategy has not been previously proposed in the literature.

Finally, the proposed DSAP algorithm avoids the computation to detect the emergence, re-occurrence, and disappearance of clusters in order to avoid adding any additional costs. It introduces the concept of time windows and dynamic threshold for the first time for an affinity propagation-based stream clustering algorithm. Every window can detect the emergence of new clusters while clusters older than the expiration time are removed. Unlike some of the other algorithms that require tuning several parameters to capture the current state of the model, DSAP uses a simple combination of autonomously calculated dynamic threshold and expiration time to keep the algorithm more efficient.

Chapter 4

Data Stream Affinity Propagation

This Chapter describes the research methodology adopted for developing, implementing, and evaluating the proposed Data Stream Affinity Propagation (DSAP) algorithm using the landmark time window model. This algorithm is based on previous research work where the online-offline clustering phases have been suggested to overcome the clustering limitations such as detecting abrupt and gradual changes as soon as they occur, and distinguishing drift from noise. Figure 4.1 provides an overview of the methodology which can be described as follows:

- **DSAP Clustering Phase:** computes micro-clusters using a landmark time window model to harvest data streams. It also computes macro-clusters by re-clustering all the centroids of the micro-clusters found in each time window. The goal is to discover hidden dense clusters structures from indoor localization data streams.

- **Intrinsic Clustering Validation Phase**: assesses the quality of the clustering results when there is no ground truth label of data. The selected metrics are silhouette index, Caliński-Harabasz index, and Davies-Bouldin index (scikit-learn developers, 2020). The focus is to assess between-clusters dispersion and inter-cluster dispersion for all clusters.

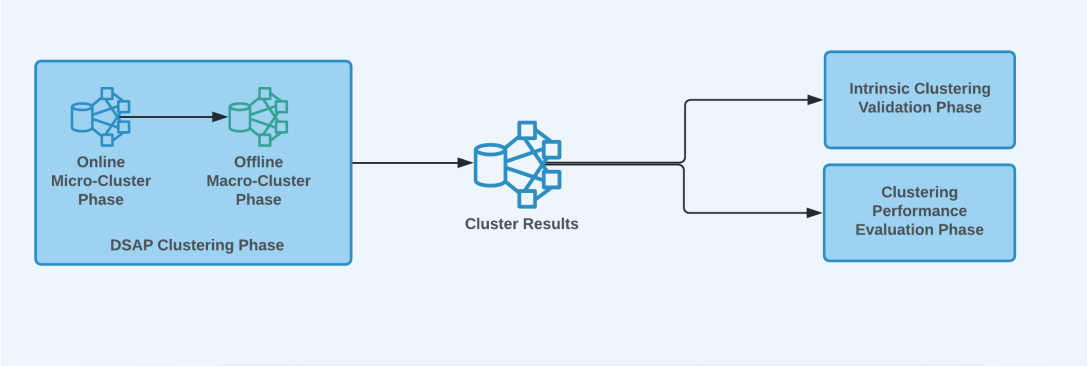


Figure 4.1: Overview of the DSAP Phases

- **Clustering Performance Evaluation Phase:** estimates the efficiency of the DSAP algorithm using the time and space complexity metrics. The aim is to find the right balance between memory consumption and the speed of execution for the DSAP algorithm.

These phases are described in more detail in the following sections. First, The DSAP clustering phase, with two phases, online micro-cluster and offline macro-cluster, is introduced. Then, the clustering validation and performance evaluation phases are explained.

4.1 The DSAP Clustering Phase

The DSAP algorithm is based on the online and offline clustering phases. The online micro-cluster phase detects newly arriving data points and updates the historical clusters accordingly. The offline macro-cluster phase generates summary clusters from the centroids of the micro-clusters.

4.1.1 Online Micro-Cluster Phase

The online micro-cluster phase initializes by receiving data streams as a continuous sequence of data points that are harvested using the landmark time window

model. Therefore, the specifications for creating a landmark time window are its start time and landmark duration L with Ω data points, where x_i is the i^{th} data point in the j^{th} window (W_j) in such a way that

$$W_j = \begin{cases} 0 & : \text{if } i < \Omega \\ x_{i+(j-1)*\Omega} & \text{, where } i\text{=1,2,...,}\Omega \ : Otherwise \end{cases} \tag{4.1}$$

Data points in the same landmark window are treated as equally important. In practice, the landmark duration can be defined by using a landmark time interval (e.g. every hour) or by a landmark event (e.g. every 100 data points). Every time a new window is created, all the data points from the previous windows are discarded as illustrated in Figure 4.2. In this thesis, the landmark time interval was selected for clustering localization data streams as for these applications only the summary of the system state within a particular window was required. The time interval value can obtain based on user requirement and data rate. This window model is suitable for our clustering process, since we are interested in the most recent data within the time frame. All data points within the window have the same weights, which make appropriate for comparing hourly, daily, or monthly summary comparisons.

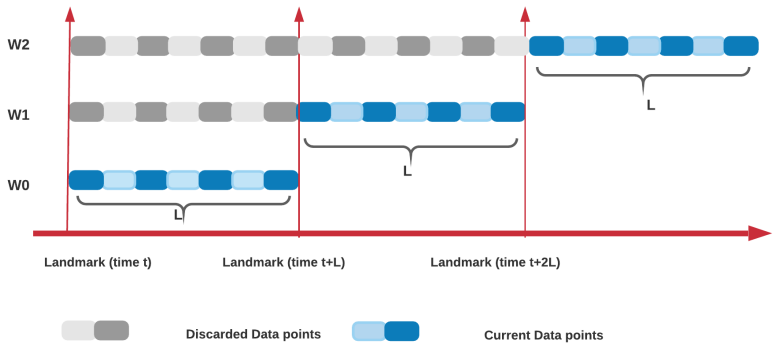


Figure 4.2: Landmark time window model

This online micro-cluster phase consists of three major steps: initialization,

comparison, and activateAP with update ϵ_{W_j}.

Initialization

The data stream is ingested in a windowed manner using the landmark time window model $W_0, W_1, ..W_j, .., W_N$, where each window W_j contains Ω data points and L is defined as the time interval of window $W_j = [x_1, x_2..., x_i, ..x_\Omega]$.

The clustering process is initiated by assigning the first sequence of data points from a data stream to the initial landmark time window W_0 as illustrated in Figure 4.2. The landmark time interval is a-priori determined based on user requirements, data rate, and expected latency of the expected data streams. For example, if high latency is expected in harvesting the data points, a long duration for the time intervals is advised. In contrast, in processing data from high rate streams, short time intervals will improve the clustering process.

The AP algorithm is applied to the all data points belonging to the first time window W_0 for computing the initial micro-clusters and their respective centroids $C_m = C_{m_1}, C_{m_2}, ..C_{m_q}$.

The clusters and data points produced by the DSAP algorithm are represented as sequence of tuples:

$$[s_1 = (C_{m_1}, N_1, t_1)], [s_2 = (C_{m_2}, N_2, t_2)],$$

$$..., \left[s_q = \left(C_{m_q}, N_q, t_q\right)\right]$$

where

- C_{m_q} is the centroid of a micro-cluster q,
- N_q is the number of data points assigned to the cluster q, and
- t_q is the last timestamp assigned to the micro-cluster q.

All data points within the initial landmark time window are considered as a potential exemplar until a robust set of centroids and their respective micro-clusters are found by computing the four AP matrices, i.e. similarity matrix, responsibility matrix, availability matrix and criterion matrix. The hyperparameters such as the preference parameter, the damping factor, the maximum number of iterations, and the convergence iteration are also set up during this phase. Their values will be constant throughout all the landmark time windows in the online micro-cluster phase. The preference value keeps as a median of the similarity matrix, and the AP algorithm automatically calculates it each time. The damping factor is in the range of 0.5 to 1, and based on data, it varies. It can be obtained by testing different values. The rest of the hyperparameters are kept as default values.

A new dynamic threshold ϵ_{W_j} is introduced in the DSAP algorithm for incrementally computing the clusters as time passes by, and as a result, allowing the analysis to take into account the changes of recurrently occurring clusters. For the initial time window W_0, an intra-cluster distance matrix is calculated between all data points and their respective centroids in each cluster using the Euclidean distance function. The mean of all these intra-cluster distances gives the initial value of the ϵ_{W_0} threshold that will be used in the next step.

The outputs of the initialization step are a set of micro-clusters centroids (C_m) and an initial ϵ_{W_0}.

Comparison

As new data points arrive, the DSAP algorithm allocates them into their respective new time windows $W_j = [x_1, x_2..., x_i, ..x_\Omega]$ using the same landmark time interval L as defined in the previous step. For each new data point, a set of tasks are devised as follows:

- Compute the Euclidean distances between each new data point x_i and the

current micro-cluster centroids C_{m_q} using Equation 4.2.

$$d\left(C_{m_q}, x\right) = \sqrt{\sum_{1}^{n} \left(x_i - C_{m_{q_i}}\right)^2} \tag{4.2}$$

where

- C_{m_q} is a current micro-cluster centroid,
- x is a new data point within the current landmark time window.
- x_i and $C_{m_{q_i}}$ are Euclidean vectors starting from the origin
- n space

- Compare the computed Euclidean distances against the current ϵ_{W_j}. If one of the computed distance values is less than the current ϵ_{W_j} threshold (i.e. $min(d([C_m], x_i)) < \epsilon_{W_j}$), this new data point will be straightforwardly placed within a current existing cluster. The DSAP algorithm merges the new data point x_i with the closest existing cluster by updating the time t_q and adding to the number of data points N_q values in the cluster centroid tuple C_{m_q}. However, if all computed distance values are higher than the current ϵ_{W_j} threshold, the new points will be saved in the time-window repository temporarily until the landmark time window ends, and later be used in the next ActivateAP and Update step.

Figure 4.3 illustrates the previously described tasks by showing a sparse and a dense micro-cluster with their respective centroids C_{m_i} and C_{m_j}. With the arival of new data points, x_i and x_j, the intra-cluster distances T and T' are computed in a time window W_1. In this case, the T distance is less than the current ϵ_{W_1}, and the new data point is merged with the existing cluster C_{m_i} (Figure 4.3a). In contrast, the computed distance T' is greater than the current the current ϵ_{W_1} threshold, and this new data point will be saved in the time-window repository until the end of the

streaming for the time window W_1 (Figure 4.3b). These tasks are performed every time a new landmark time window is created and new data points arrive.

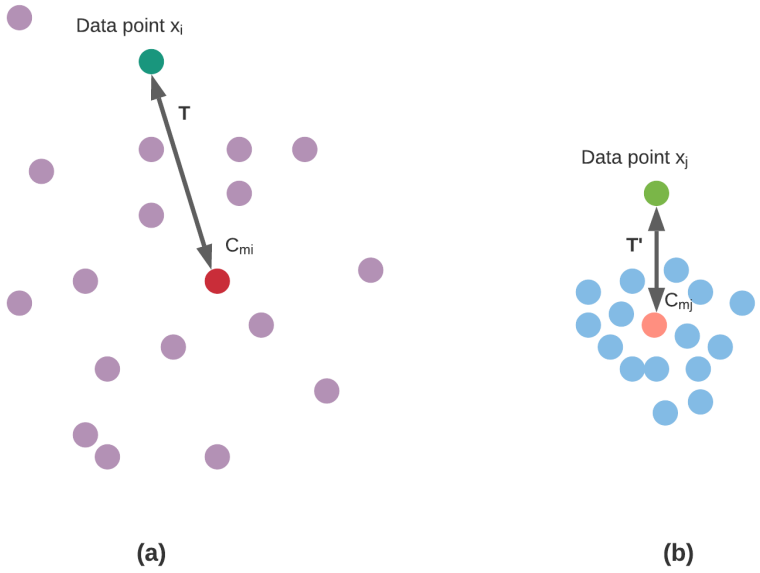


Figure 4.3: The impact of spread (a) and dense (b) cluster structures on comparing the closeness of new data points to existing micro-cluster centroids C_m

The outputs of the comparison step are the set of micro-clusters from the previous window, but now with new data points associated with some of the centroids, and an updated timestamp corresponding to the chosen centroids. Additionally, n number of data points whose Euclidean distance from the centroids was greater than ϵ_{W_j} are temporarily kept in the time-window repository.

ActivateAP and Update ϵ_{W_j}

All the data points that have been temporarily accumulated during the execution of the previous step are used to compute new micro-clusters C'_m using the AP algorithm once again. These new clusters will be added to the existing clusters C_m that belong to their respective landmark time windows. As a result, the updated $C_m = C'_m + C_m$ is achieved. The new micro-clusters will have their respective

timestamps associated with them.

The next task is to compute the new ϵ_{W_j} by taking into account the centroids of the new micro-clusters found within a particular landmark window. This is accomplished by computing the mean of the intra-cluster distances using the new micro-clusters. The update ϵ_{W_j} is computed by applying the mean between the new and the current ϵ_{W_j}. The updated ϵ_{W_j} is then used when the forthcoming data points from a new landmark time window arrive.

In order to avoid the number of centroids spreading beyond control, old centroids that are no longer deemed relevant are removed. To do this, an expiration time hyperparameter e_x is introduced. This value is applied to forget centroids that have not been selected in the recent windows as quantified by e_x. The e_x number can be chosen to include all the time windows from the start or just the last few windows, depending on the user requirements. All the clusters whose associated timestamps are older than e_x window lengths ($e_x \times L$) from the current time are considered obsolete and hence discarded.

4.1.2 Offline Macro-Cluster Phase

The offline macro-cluster phase in DSAP starts after N time windows have passed. In this phase, all the micro-clusters C_m generated during the online micro-cluster phase are re-clustered using the AP algorithm to generate k number of macro-clusters $C_M = (C_{M_1}, C_{M_2}, ..., C_{M_k})$. This process is usually not considered time critical, and the number of macro-clusters is expected to be less than the number of micro-clusters. The hyperparameters related to the AP algorithm, such as the maximum number of iterations and the convergence iteration, are default values. The preference parameter is set to median the similarity matrix of all the micro-clusters obtained from the online phase, and the damping factor is a fixed number for this phase.

4.2 Intrinsic Clustering Validation Phase

Intrinsic clustering validation uses the internal information of the clustering process to evaluate the goodness of a clustering structure when the ground truth labels are unknown. The well-known metrics silhouette index (S), Caliński-Harabasz index (CH), and Davies-Bouldin index (DBI) were selected to validate the goodness of macro and macro clusters found by the DSAP algorithm. Table 4.1 provides an overview of these metrics. They are further described in detail in the following sections.

Table 4.1: Intrinsic clustering metrics used for validation of clustering results.

Metrics	**Description**	**Advantages**	**Disadvantages**
Silhouette Index	It measures how similar a data point is to its own cluster compared to other clusters.	* Value bounded between [-1,1] * High scores when clusters are well separated	* High computational complexity * Higher value for convex clusters unlike density-based clusters
Caliński-Harabasz Index	It is the ratio between the intra-cluster dispersion and the inter-cluster dispersion.	* Fast to compute * Easy to interpret	* Higher value for convex clusters
Davies-Bouldin Index	It is the average similarity measure of each cluster with its most similar cluster.	*Simple to implement *Value is able to compute when features inherent to the dataset	* Higher values for convex clusters * Limited to Euclidean distance only

4.2.1 Silhouette Index

The silhouette index was introduced by (Rousseeuw, 1987) on the premise of the silhouette width of a data point to measure how similar a data point is to its own cluster compared to other clusters. The silhouette index S_i for a data point $i \in x$ in cluster $C_k \in C$ is calculated using Equation 4.5.

$$a_i = \frac{1}{C_k} \sum_{j \in C_k, i \neq j} d(i,j) \tag{4.3}$$

$$b_i = \min_{C_m \in C, C_m \neq C_k} \frac{1}{|C_m|} \sum_{j \in C_m, i \neq j} d(i,j) \tag{4.4}$$

$$s_i = \frac{b_i - a_i}{\max(a_i, b_i)} \tag{4.5}$$

The first parameter a is the average distance between the sample and all the others in the same class, and the second parameter b is the mean distance between the sample and all the other points in the next closest cluster. Negative S_i scores for a point i means that the data point is in the incorrect cluster. Positive scores show a robust and dense clustering. Moreover, values around zero indicate overlapping clusters. If the S_i has a higher value, it means a greater ratio between b_i as compared with a_i, causing the data point i to be more similar to the other data points in the same cluster.

4.2.2 Caliński-Harabasz Index

It is also known as the variance ratio criterion, which is used to score dense and well-separated clusters. A recent comparative study of available clustering indices demonstrated this index as one of the best cluster validity indices (Arbelaitz, Gurrutxaga, Muguerza, Pérez, & Perona, 2013).

Well defined clusters yield high values of this index. Therefore, the maximum value of the index is used to select the best partition. For n data points, k clusters where B and W are the between within cluster scatter matrices, the index is computed as (Caliński & Harabasz, 1974):

$$CH = \frac{traceB/(k-1)}{traceW/(n-k)} \tag{4.6}$$

where

$$B_k = \sum_{q=1}^{k} n_q (c_q - c_i)(c_q - c_i)^T \tag{4.7}$$

$$W_k = \sum_{q=1}^{k} \sum_{x \in C_q} (x - c_q)(x - c_q)^T \tag{4.8}$$

with C_q the set of points in cluster q, c_q the center of cluster q, c_D the center of D, and n_q the number of points in cluster q.

The CH is computationally efficient to calculate, but just like the silhouette index, the CH index is biased towards convex clusters, assigning them higher values than the density-based clusters such as those obtained through DBSCAN.

4.2.3 Davies-Bouldin Index

The Davies-Bouldin Index (DBI) was introduced in 1979 by (Davies & Bouldin, 1979), and it computes the average similarity between each cluster C_i for $i = 1, 2, ..., k$ and its most similar one C_j. For this index, similarity is defined as a measure R_{ij} that optimizes:

- s_i: the average distance between each data point in the cluster and the centroid of that cluster.
- d_{ij}: the distance between cluster centroids i and j.

A simple choice to construct R_{ij} is that of non-negative and symmetric as follows:

$$R_{ij} = \frac{s_i + s_j}{d_{ij}} \tag{4.9}$$

Then the Davies-Bouldin index is defined as:

$$DB = \frac{1}{k} \sum_{i=1}^{k} \max_{i \neq j} R_{ij} \tag{4.10}$$

With this index, a minimum value denotes the best partitioning of the data. The average similarity calculation is much simpler than the computations required for calculating the silhouette index. Like the previous two metrics, DBI too suffers from a preference towards convex clusters as compared to density based clusters owing to the usage of centroid distance that limits the distance metric to Euclidean space.

4.3 Clustering Performance Evaluation Phase

Data stream clustering brings about a few unique challenges as compared to traditional data clustering. The data streams are a continuous flow of data points that arrive at a rapid rate. Random access to these data points is not possible and the volatility of the data streams sometimes limits to having a single look at the data points upon their arrival.

To evaluate the efficiency of the DSAP, two metrics are proposed: computational time and the memory consumed to run the DSAP algorithm. These metrics used in this phase are described in the next sections.

4.3.1 Time Complexity

The efficiency of any algorithm can be estimated using the time complexity, which can be defined as the time taken to run the number of elementary operations performed by the algorithm given an input array of length N. In other words, it is the amount of time taken by an algorithm to run, as a function of the length of the input.

The most frequently performed action within the algorithm whose execution time does not change with the size of the input array is called the elementary operation. The most common metric used for estimating time complexity is the Big O

notation. The big O notation is typically used to illustrate the worst-case growth of an algorithm (or how the run time/space requirements of an algorithm increase with input size), which excludes coefficients and lower order terms.

The AP algorithm has a quadratic computational complexity of $O(N^2 logN)$ for input length N forbidding the use of AP on large data sets (Frey, 2007). The time complexity of the DSAP algorithm is affected by three steps in online phase and one step in offline phase. For a data set size N, divided into time windows of size L, the complexity for the initialization step would be $O(L^2 logL) + O(L)$, for AP and threshold calculation on L points. In comparison step, Euclidean distances are calculated for the subsequent L point in the next time window leading to complexity of $O(L)$. If the time window repository has T data points, then the activateAP step will have a time complexity of $O(T^2 logT) + O(T)$ for the second instance of AP and threshold calculation. Finally C_m micro clusters are clustered by AP whose time complexity is $O(C_m^2 logC_m)$. As a result, total worst case complexity of the algorithm is $O(L^2 logL) + O(L) + N/L(O(L) + O(T^2 logT) + O(T)) + O(C_m^2 logC_m)$, which is affected by the initial window size, the number of points in the repository and the number of micro clusters. It is expected that for highly dynamic data set, the execution time would be higher since more number of micro-clusters will be generated.

In real-world applications, in addition to the time complexity, several other factors can affect the execution speed. Many concurrent tasks like receiving network traffic, checking the hard disk, and writing log files, are performed by a typical computer while it performs the clustering tasks. The data-dependent rate of convergence of the algorithms also affects the total execution time. Therefore, it is recommended to run a task multiple times to get average values after discarding the initial few results as they could be affected by the time-window repository results.

4.3.2 Space Complexity

Similar to the time complexity, the memory consumption quantifies the amount of memory taken by an algorithm to run as a function of the length of the input. That means how much memory, in the worst case, is needed at any point by the algorithm. The same big-O notation is used to describe the space complexity as well. This parameter represents the algorithm's scalability and performance. In simple terms, it gives the worst-case scenario of an algorithm's growth rate.

To calculate space complexity, we need to know the value of memory used by different types of datatype variables, which generally vary for different operating systems. Hence it is not straightforward to provide generic formulations for the spatial complexity of any algorithm.

To compare different algorithms, we have to make sure that they have been suitably implemented since poor memory management can lead to a superior algorithm under-performing as compared with the superior implementation of a less efficient algorithm. Analogous to the time metric, it is necessary to test the algorithms on data sets of multiple sizes for an accurate assessment of the memory usage.

4.4 Summary

Figure 4.4 summarizes the entire research methodology discussed in this chapter. The four phases of the DSAP approach are depicted with their main steps and respective tasks.

The online micro-cluster and offline macro-cluster phases are depicted within the grey box in blue and green colors respectively in the flow chart shown in Figure 4.4. The online phase initializes the computation of micro-clusters and computes the updated micro-clusters from the new data points harvested using the landmark time window. The offline phase generates macro-clusters, a summary of all data points from the stream, by re-clustering the micro-clusters.

The silhouette index, Caliński-Harabasz index, and Davies-Bould index are proposed to assess the goodness of the clustering structures of the discovered micro and macro-clusters since the ground truth labels were unknown. This phase is illustrated in Figure 4.4 in bright red color. The efficacy of any stream clustering algorithm is tied to its memory consumption and speed of execution. In order to evaluate this, in the Clustering Performance Evaluation phase shown in Figure 4.4 in purple color, the time and space complexity of the DSAP algorithm are estimated.

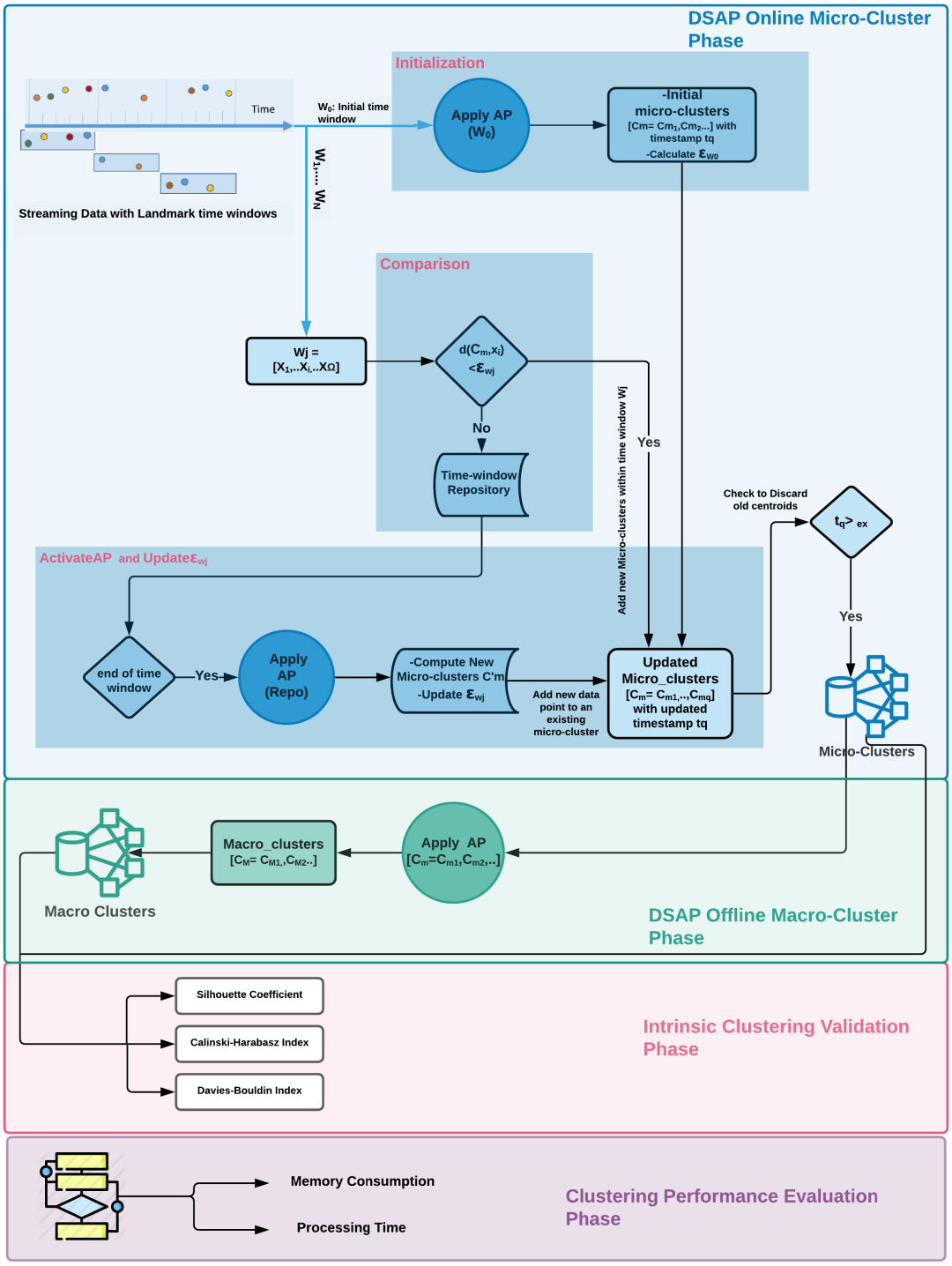


Figure 4.4: Flowchart of the proposed DSAP approach

Chapter 5

Implementation

In this chapter, the design and implementation of the DSAP algorithm are described. Next, two experiments are presented that are related to applying the proposed DSAP model to find stair usage patterns and occupant behaviour in indoor spaces.

5.1 Data Stream Simulation

In order to mimic a data stream, it was necessary to convert the collected data into a stream. For DSAP, this conversion was performed using the scikit-multiflow data stream simulation framework from Python. Scikit-multiflow is a free and open-source tool and it is part of the River machine learning library in Python (Montiel, Read, Bifet, & Abdessalem, 2018). The prerequisites to install scikit-multiflow package (*skmultiflow*) are NumPy and Cython libraries.

The `skmultiflow.data` module includes two group of classes that either perform batch-to-stream conversions or generate standard data streams like waveform, Gaussian as shown in Figure 5.1. The batch-to-stream conversion classes generates streams from a file source (.csv files) in the disk or from data in the memory.

The DSAP uses the *skmultiflow.data.FileStream* class for generating data stream

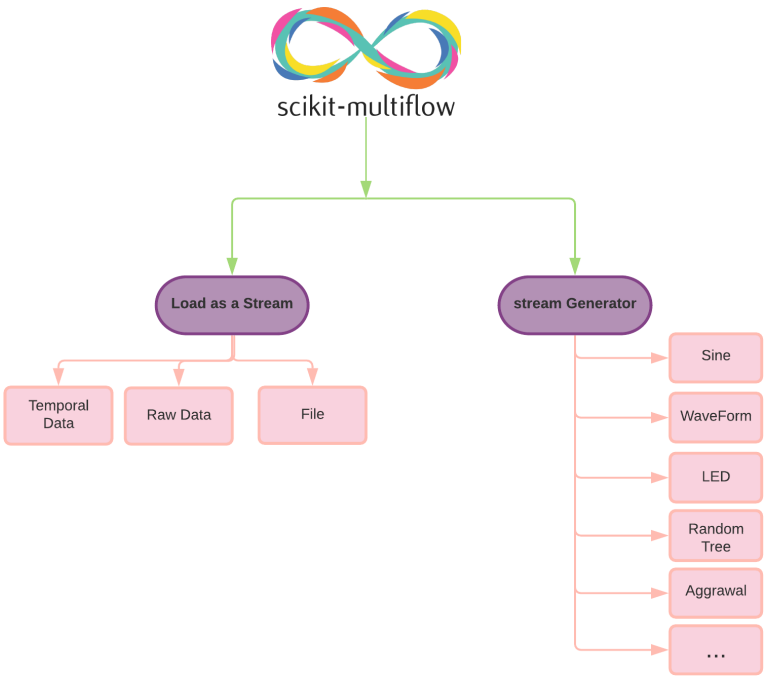


Figure 5.1: The scikit-multiflow data stream generator modules: load as a stream and stream generator

from any `CSV` file. It is part of the `skmultiflow.data` module and imported as: `skmultiflow.data.file_stream`. Then, the data set is loaded as a stream (`stream = FileStream (filepath)`), which is generated by the FileStream class. The Stream class is in charge of providing data inside scikit-multiflow and it runs upon request a number of samples, in a way such that old samples cannot be accessed at a later time.

The most important method of the stream class is `next_sample()`, which returns the next window at the start of every time interval. Also, the function `has_more_samples()` checks if any data point is left in the data set for consideration. Even though these packages are very capable of mimicking the data observed in data stream, the user still must acknowledge the limitations of such techniques. The difference from the real data stream can result from many potential shortcomings of the stream simulation models. The stream simulation packages do not have

the support for generating realistic latency information, missing data, mixed timestamps, varying data cadence, and others. Hence any evaluation of the robustness and efficacy of a stream clustering algorithm done using the simulated streams should be cautiously accepted.

5.2 DSAP Clustering Phase

The DSAP source code in Python programming language has been made freely available at: `https://github.com/nasrineshraghi/DSAP`. The detailed description of each section of the code is presented below.

After the conversion of the CSV file data to a simulated stream, a landmark time window function `getWindowSize(start_time, L)` is applied on the data points. This function finds the number of data points in a particular time window and returns this value.

```
# Estimate the number of data points within a time window
def getWindowSize(start_time,L):
    try:
        idx1 = np.where(df[:,2] >= start_time )[0][0]
        idx2 = np.where(df[:,2] >= start_time + L )[0][0]
    except:
        idx1 = 0
        idx2 = 0
    XOmega = idx2 - idx1
# XOmega is window size
    windowSize = XOmega
    return windowSize
```

Every time window has a different start time, and this start time should be

updated after the end of the window length is reached (`start_time = start_time + L`). If there is no data point available in a particular window or the number of them are few, the scikit-multiflow function (`stream.next_sample(windowSize)`) skips that window and goes to the next one.

```
from skmultiflow.data.file_stream import FileStream
# skip if the window is empty or few number of points are available
# For example 5 data points or less
        if windowSize < 5:
            X = stream.next_sample()
            continue
        else:
            j = j +1
```

It should be noted that if the landmark window is a count-based window instead of time, `windowSize` needs to be defined as the number of data point in each window instead of getting the value from the `getWindowSize` function above. For instance `windowSize = 100` means that each time the latest 100 data point from the stream are accessed. In this case number of data points in the window Ω is equal to L.

The DSAP online micro-cluster phase is implemented in three steps as described before in chapter 4: initialization, comparison, and ActivateAP with update ϵ_{W_j}.

In the Initialization step, the AP algorithm is applied on all data points present in the first window W_0 from the data stream.

```
#(1) Initialization. AP on the first time window
from sklearn.cluster import AffinityPropagation
# AP library needs to be added first
preference = np.median(euclidean_distances(W0, squared=True))
AP = AffinityPropagation(preference, damping).fit(W0)
cluster_centers_indices = AP.cluster_centers_indices_
```

```
labels = AP.labels_
Cm = AP.cluster_centers_
n_clusters_ = len(cluster_centers_indices)
X = np.array(xi)
X_label = np.array(labels)
```

The AP algorithm generates a set of micro clusters (C_m). To initiate the DSAP model, a sufficient number of data points is required in the first window. Hence, it is important to choose an optimal value for the window size; otherwise, DSAP cannot reach convergence, and the model will fail. After the micro-clusters are generated, the distance matrix of all data points and centroids in each cluster are calculated by the `getAdaptiveThreshold()` function. The average distance of these values gives the initial ϵ_{W_0} that will be used in the comparison step.

```
from scipy.spatial import distance
# Calculate the threshold distance
def getAdaptiveThreshold(Wj,Cm):
    epsilon = np.mean(distance.cdist(Wj,Cm,'Euclidean'))
    return epsilon
```

In addition, the timestamp associated with each cluster centroid should be kept in an array with the size of initial centroids. The times for all centroids in the first window are equal.

```
#time associated to initial centroids
tq = np.ones(len(Cm))*start_time
tq = tq.astype(int)
```

The output of the initialization step are a set of micro-clusters with centroids (C_m) and an initial threshold value (ϵ_{W_0}).

The next step of the online micro-cluster phase is comparison. For the following windows from W_1 to W_N, all data points in each window are compared with the existing micro-clusters. The Euclidean distance of each new point x_i in the window to its closest micro-cluster centroid is calculated. If the distance is less than the threshold ϵ_{W_j}, x_i is added to the micro-clusters and the timestamp associated with those specific centroids needs to be updated. Otherwise, it is kept temporarily in the time-window repository until all data points in the window are evaluated.

```
#(2) Comparison step.
# Update the start time for each window
windowsize = getWindowSize(start_time, L)
start_time = start_time + L
indx = 0
repository = []
for i in range(windowSize):
    d = distance.cdist(Cm,x,'Euclidean')
    ind = np.argmin(d)
    if min(d) < epsilon:
         X = np.concatenate((X , np.reshape(x[i,:],(-1,2))))
         X_label = np.concatenate((X_label , np.reshape(ind,(-1))))
         #Updating time associated to centroid met in this step
         tq[ind] = start_time
    else:
        repository.append(x[i])
        indx = indx +1
```

In the next ActivateAP and update ϵ_{Wj} step, AP is applied on all the remaining data points in the time-window repository whose distances from the centroids are more than the adaptive threshold ϵ_{W_j} for the current time window. In the end,

clusters obtained from this step are added to the updated micro-clusters.

```
#(3) ActivateAP step for the time-window repository points
repositorydata = np.array(repository)
preference = np.median(euclidean_distances(repositorydata, squared=True))
AP_crepository = AffinityPropagation(preference,
    damping).fit(crepositorydata)
C‘m = AP_repository.cluster_centers_
AP_repository_label = AP_repository.labels_
X = np.concatenate((X , np.reshape(repositorydata[i,:],(-1,2))))
X_label = np.concatenate((X_label , np.reshape(AP_repository_label,(-1))))
#C‘m is the Cm prime, new micro-clusters
Cm = np.append(C‘m, Cm, axis = 0)
# Assign timestamp to new micro-clusters
repository_tq = np.ones(len(C‘m))*start_time
repository_tq = repository_tq.astype(int)
tq = np.concatenate((tq, repository_tq), axis=0)
```

As new micro-clusters are generated at the end of the ActivateAP step, the threshold value needs to be recalculated based on the updated micro-clusters. Once again the adaptive threshold function is called upon to estimate new threshold values. This value is compared with the threshold ϵ_{W_j}. If the value is greater than ϵ_{W_j} then the average of the two thresholds is selected as the new threshold; otherwise the threshold remains unchanged. The decision to choose the average of the two numbers was taken to ensure that the threshold does not blow up in size due to a sparse cluster or noise.

```
# Update the threshold value.
epsilonC‘m = getAdaptiveThreshold(C‘m, repositorydata)
if epsilon < epsilonC‘m:
```

```
    epsilon = (epsilonC‘m + epsilon)/2
```

In order to keep the number of micro-clusters under control and avoid obsolete clusters, cluster centroids that are older than the expiration number e_x (older than a specific number of landmark) are removed periodically. The oldest relevant time is calculated by subtracting the e_x number of time periods from the current timestamp (`start_time`). The centroids with timestamps greater than this value are retained in the updated centroids along with their updated associated timestamps. Based on the duration of interest, this expiration value is set by the user at the start of the stream clustering process.

```
# Remove old micro-clusters
# Number is an integer value defines by the user
ex = int Number
remain_centers = np.ones(len(tq))
for i in range(len(tq)):
    # Remove the centroids with time more than the ex
    if tq[i] < start_time - ex * L:
        remain_centers[i] = 0
indx_remain= np.where(remain_centers==0)
Cm  = Cm[remain_centers==1]
tq = tq[remain_centers==1]
a = indx_remain[0]
for j in range(len(a)):
    indx = np.where(X_label == a[j])
    # Remove the labels with time more than ex
    X_label= np.delete(X_label, indx)
```

The next phase of the DSAP algorithm is the offline macro-cluster phase which generates final clusters by applying the AP algorithm one more time to all the micro-

cluster centroids accumulated during the online phase (`AffinityPropagation().fit(`Cm`)`). In the offline phase only the summary statistics represented by the micro-clusters centroids stored during the online phase are used and not the actual data set itself. The number of micro-clusters is a lot smaller than the total number of points in the data set that it represents. The number of macro-clusters obtained in this phase is less than the number of micro-clusters as they summarize the whole period of observation and not individual windows. The pseudo-code presented in Algorithm 2 shows the outline of the entire DSAP algorithm.

```
# Offline Macro-clustering Phase
preference = np.median(euclidean_distances(Cm, squared=True))
AP_mac = AffinityPropagation(preference, damping).fit(Cm)
cluster_centers_indices_mac = AP_mac.cluster_centers_indices_
cluster_centers_mac = AP_mac.cluster_centers_
mac_label = AP_mac.labels_
```

5.3 Intrinsic Clustering Validation Phase

To validate the clustering results, three Intrinsic evaluation metrics: Silhouette Coefficient, Caliński-Harabasz Index, and Davies-Bouldin Index, are used as discussed earlier in section 4.2. All three metrics were implemented in Python.

- *Silhouette Index:* This cluster evaluation metric was implemented as a function `silhouette_score` and is available in the `sklearn.metrics` library in Python. The input parameter of this function is an array of pairwise Euclidean distances between micro-cluster centroids and the final macro-cluster centroids,

- *Caliński-Harabasz Index:* It is available in the `sklearn.metrics` library of Python as the `calinski_harabasz_score` function. The input parameter of

Algorithm 2: DSAP Algorithm

Input: Data stream: S as windows $W_0, W_1, ..., W_N : W_j = x_1, ..., x_\Omega$ with the length L, ex
Hyperparameters: Preference, Damping, ϵ_{W_j}
Output: Set of micro clusters with centroids $Cm = [Cm_1, Cm_2, ..., Cm_q]$, set of macro clusters with centroids $C_M = [C_{M_1}, C_{M_2}, ...C_{M_k}]$

Function `getWindowSize`$(start\_time, L)$
  $time\_period = start\_time + L$
  $windowSize = S(time\_period)$
Result: $windowSize$

Function `getAdaptiveThreshold`$(W_j,\ Cm)$
  $\epsilon_{W_j} = \text{mean}(\text{d}(W_j, Cm))$
Result: ϵ_{W_j}

Online Phase

j = 0
while *!(stream.has_more_samples())* **do**
  if *(j == 1)* **then**
    ▷ Initialization
    $W_0 =$ `S.getWindowSize`$(start\_time, L)$
    Function `AP(` W_0`)`
    Result: Initial micro-clusters: $Cm = [Cm_1, Cm_2, ..., Cmq]$, label data points: X, $[tq] = start\_time$
    $\epsilon_{W_j} = \text{getAdaptiveThreshold}(W_0, Cm)$
  else
    $W_j =$ `S.getWindowSize`$(start\_time + L * j, L)$
    $start\_time = start\_time + L$
    for *($x_i : i = 1 : \Omega$)* **do**
      ▷ Comparison
      $[d] = \text{d}([Cm], x_i)$
      if *min ($[d]$)* $< \epsilon_{W_j}$ **then**
        $Cm[X] \leftarrow x_i$ (merge with micro_clusters)
        $[tq] = start\_time$
      else
        $repository \leftarrow x_i$
      ▷ ActivateAP
      Function `AP(`$repository$`)`
      Result: new micro-clusters:$[C'm]$
      $\epsilon'_{W_J} = \text{getAdaptiveThreshold}(C'm, repository)$
      $tq_{repository} = start\_time$
    Update micro_clusters: $Cm = Cm + C'm$
    Updated timestamp $tq = tq + tq_{repository}$
    Update Threshold : ϵ_{W_j}
  if $tq < start\_time - ex * L$ **then**
    Discard Cm
  Results: Micro Clusters: $Cm = [Cm_1, Cm_2, Cm_3, ..., Cm_q]$

if *end of S* **then**

Offline Phase

Function `AP(`Cm`):`

Result: Macro Clusters: $C_M = [C_{M_1}, C_{M_2}, ...C_{M_k}]$

this metric is a list of n_features-dimensional data points in which each row corresponds to a single data point and labels related to each data. The output of this function is a float number with no cut-off value. The higher the value, the better is the solution.

- *Davies-Bouldin Index:* The last metric is the Davies-Bouldin Index that was implemented in Python as part of the same library `sklearn.metrics` as CHI. The function, which is called `davies_bouldin_score`, has two input parameters: list of n_features-dimensional data points and labels. This result is a float number whose lower values indicate tighter clusters that are better separated.

5.4 Clustering Performance Evaluation Phase

Analogous to the intrinsic cluster validation phase where the quality of the clusters was assessed, in this phase the DSAP algorithm is evaluated in terms of computational time efficiency and memory consumption. Straightforward functions are available in Python to compute these two metrics.

- *Processing Time:* To find the running time in Python, the time function within the `time()` module is used to get the execution time of the algorithm. This module calculates the running time of a program by first storing the starting time and ending time, before the first and after the last line. At the end, the difference between these two times would be the total running time of the program. The `time()` function returns the number of seconds elapsed since the epoch.

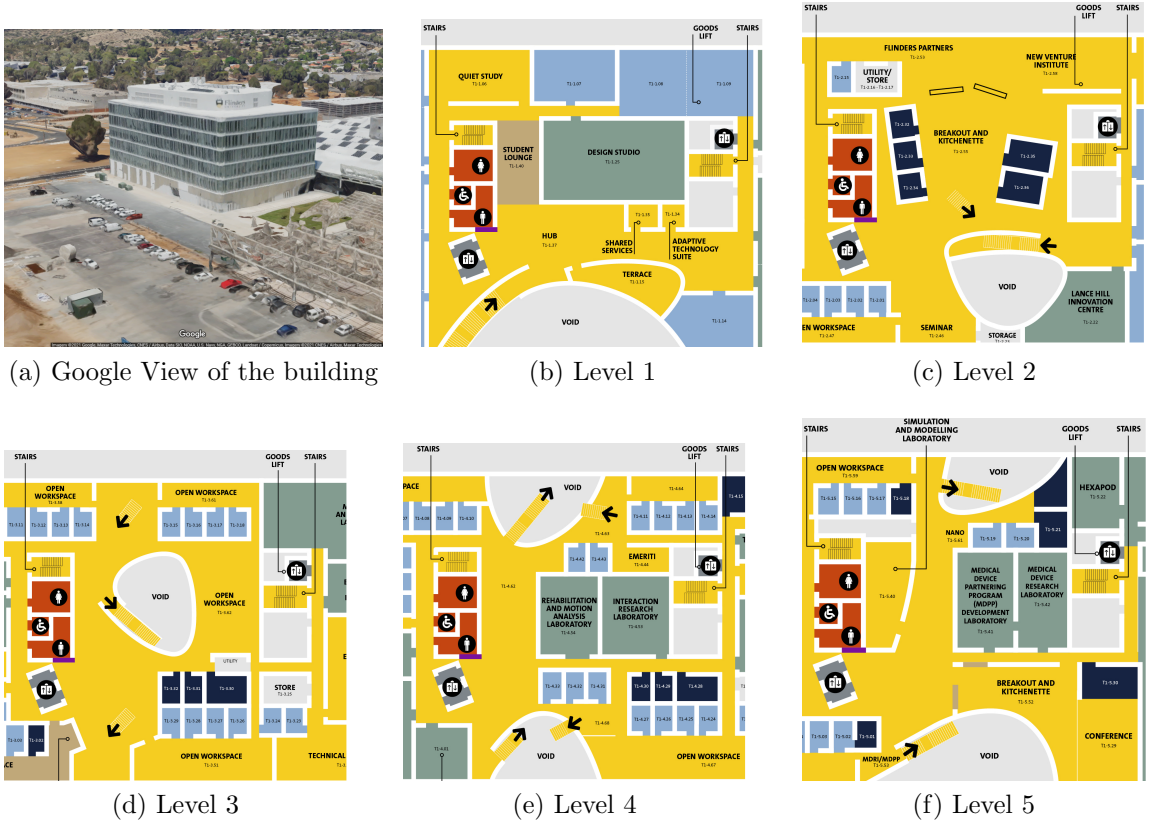
- *Memory Consumption:* To monitor memory usage, the `os` and `psutil` libraries in Python are used. Psutil (Python system and process utilities) is a cross-platform library for retrieving information on running processes and system utilization (CPU, memory, disks, network, sensors). `psutil.virtual_memory()` function returns statistics about system memory usage. Also, Python provides a C module called cProfile, which provides deterministic profiling of Python programs. A profile is a set of statistics that describes how often and for how long various parts of the program are executed. This module is mainly used when we wanted to calculate the window length and threshold.

5.5 Behavioural Intervention Experiment

Our first case study consists of an experiment performed in the Tonsley building at the campus of Flinders University, Australia in 2019. This study was conducted to observe the potential impact of motivational and educational interventions on the physical activity of people with sedentary lifestyles.

Tonsley Building is a six-story building populated by students and staff with entrances facing north and south directions. The building floor layouts are illustrated in Figure 5.2. The north-side entrance faces a small barricaded entrance as seen in panel 5.2a, but the main entrance is on the south side adjacent to the covered Tonsley garden. The building is fitted with lifts that reach all the levels and have stairs on the north and south sides that connect all the levels. Additionally, the building has an open space called the void and has central stairs that connect all the levels. Thus, each level, stairs are labeled as level $L_{x+1} - L_x$ North/South/Central, where $L_x = 1, 2, .5$.

E-counter sensors were deployed at all stairs. They are made by the International Road Dynamics company and consist of a pair of transmitter and re-



(a) Google View of the building (b) Level 1 (c) Level 2

(d) Level 3 (e) Level 4 (f) Level 5

Figure 5.2: Tonsley Building Layout. (a) Tonsley building view from the north-side parking lot, (b)-(f) Layout of levels 1-5

ceiver(transmitter: PTx20-1 and receiver: PRx20W1). The transmitter sends an infrared message to the receiver; when the infrared light is interrupted by someone passing through it, a count is registered. If no one passes, then these sensors log a count value of zero every 10 minutes. Each stair has two sensors, UP/DOWN that determine a person's direction of travel up and down the stairs. Signals from the sensors are sent to wireless hubs in real-time and then from the hubs to the server. The specification of all the sensors from this company are shown in Table 5.1. The maximum count value for each direction is about 1,000,000.

The data were collected from February 21, 2019, until July 16, 2019, but only three months of this data set were monitored for the experiment. The experiment was conducted for a period of three months from March 18 to June 23, 2019, as

Table 5.1: E-counter sensors of International Road Dynamics company

Features	**Transmitter**	**Receiver**	**Applications**
Display/Wireless/Bidirectional	PTx20-1	PRx20WD1	Entrance and daily counts
Wireless/Bidirectional	PTx20-1	PRx20W1	Entrance and daily counts
Display/Bidirectional	PTx20-1	PRx20D1	Entrance and daily counts
USB/Bidirectional	PTx20-1	PRx20U2	Entrance and daily counts
Display/USB/Bidirectional	PTx20-1	PRx20UD2	Entrance and daily counts

shown in Table 5.2. During the first month period starting on March 18, 2019, data were collected to establish a baseline for staircase usage by making measurements as unobtrusively as possible. The mid-semester break from April 15-28 was excluded from the study.

Motivational and educational interventions were made during the next month starting from April 29, 2019 to increase stair usage. The interventions included digital screens that displayed health information and provided live feedback throughout the building, especially near the lifts. During an additional one-month period starting on May 27, 2019, activity was observed in order to assess any long term changes in the stair use pattern after the end of the intervention period. To keep the stair usage experiment duration separate from the rest of the data, it is saved in a different .CSV file to be used for pre-processing.

The sensors captured movement for all 24 hours and all days continuously. The raw e-counter data set consists of approximately 668,000 data points, each containing nine attributes as listed in Table 5.3 out of which almost half the data points were included in our research work.

During pre-processing, some observation were detected during the experiment that affected the quality of our model. For instance, hubs were temporarily switched off or sensors fell off. Also, some special events took place during the experiment, such as public holidays, term breaks, and fire drills which affected the number of people using the stairs. For example, on May 1, at 11:15 a.m, the fire alarm went on,

Table 5.2: Experiment timeline at Tensely Building for three months of intervention. The green color shows the before intervention month, the blue color indicates the intervention month, and the yellow color represents the month after the intervention period. The red and pink colors represented anomalous periods and are out of scope for this experiment.

Semester Weeks	Timeline	Date
week 12	Baseline(Before intervention)	3/18/2019
week 13	Baseline	3/25/2019
week 14	Baseline	4/1/2019
week 15	Baseline	4/8/2019
	Mid Semester Break	4/15/2019
	Mid Semester Break	4/22/2019
week 18	Intervention	4/29/2019
week 19	Intervention	5/6/2019
week 20	Intervention	5/13/2019
week 21	Intervention	5/20/2019
week 22	Follow-up(After Intervention)	5/27/2019
week 23	Follow-up	6/3/2019
week 24	Follow-up	6/10/2019
week 25	Follow-up	6/17/2019
	Exams	6/24/2019
	Exams	7/1/2019
	Semester Break	7/8/2019
	Semester Break	7/15/2019
	Semester Break	7/22/2019

and everyone had to evacuate the building using the stairs. Also, between baseline duration and initiation of intervention, there was a two-week Easter break or semester break and that is why the intervention started after two weeks of the first stage.

Furthermore, if no one passes through the beam of sensors, the sensors indicate zero during that specific interval. A huge number of zeros were logged during the experiment especially after working hours, weekends, and holidays, which could bias the clustering results. These zero values were removed from the data set.

The e-counters save the position values as a string character combination of numbers and strings. Hence, it was also necessary to convert the string format of

Table 5.3: List of attributes for the e-counter data set

Attribute	**Measurement**	**Definition**
Date	mm/dd/yyyy	Date of measurement
Time	hh:mm:ss 24H	Time of measurement
Count	integer	Number of people counted by the sensor
Sensor	Up, Down	Direction of the sensor
Position	string	Sensor location in the building
Location Code	H2-L3	Code for each location

the position values into the integer format in order to allow us to use them as an input for the DSAP algorithm.

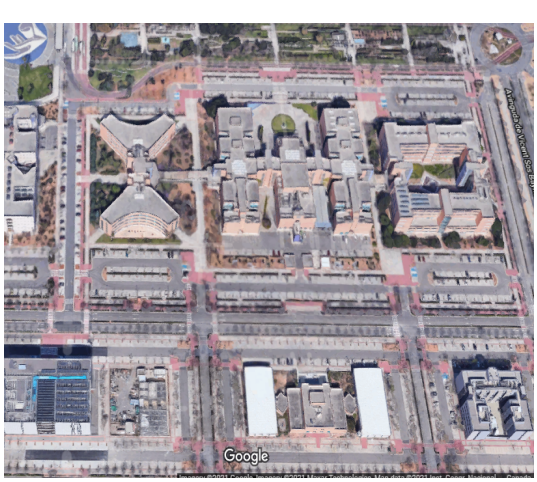
Finally, the feature selection task was performed to provide a target data set ready to be used by the simulation and clustering. The position and count features were selected for computing the clusters. The hyperparameters for the AP in the DSAP algorithm are consistent and fixed for the entire experiment. The damping value is equal to 0.96, maximum iteration is 100, and preference is equal to the median of the similarity matrix.

5.6 Occupant Behaviour Experiment

This case study was carried out to find occupant behavior patterns in indoor spaces to provide relevant information for future planning building automation, evaluating energy efficiency scenarios, and simulating emergency protocols. The experiment was conducted over one months at the Universitat Jaume I campus, in Valencia, Spain. During this time, users carried mobile devices that used *CaptureLoc* to record eight types of attributes including user position and WiFi signal strength across a total of thirteen levels spanning three buildings of the campus. To protect the identity

of the participants during the course of the experiment, the unique MAC identifiers of the phones were hashed (anonymized) on reception.

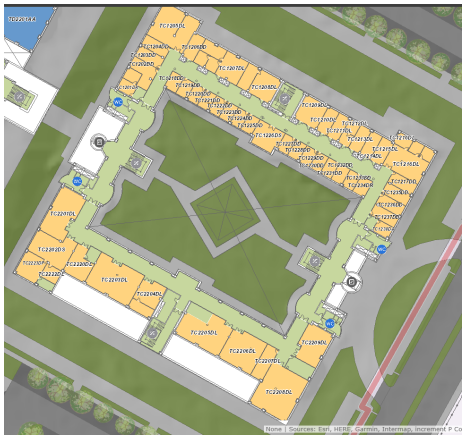
The UJIIndoorLoc data set generated includes three buildings with four or more floors (one building has five floors) and almost 110.000 $m^2$of space. The total number of locations captured was 933. The shape and structure of the three buildings TI, TD, and TC, are quite different, as shown in Figure 5.3. The raw data set consists of 19,938 data points containing 529 attributes as illustrated in Table 5.4.



(a) Overview of the three buildings

(b) Building Number 0 (TI)

(c) Building Number 1 (TD)



(d) Building Number 2 (TC)

Figure 5.3: (a) Map of the UJI Riu Sec Campus and the buildings used for WiFi fingerprint experiment. (b,c,d) The layout of the three buildings with spaceid numbers for a floor.

The data were collected From May 30 to June 20, 2013. The data collection

Table 5.4: List of attributes of the UJIIndoorLoc data set.

Attributes	**Measurement**	**Definition**
RSSI Level	-100dBM - 100dBM	WAP identifiers are linked to the vector positions
Longitude	UTM from WGS84	Y coordinates
Latitude	UTM from WGS84	X coordinates
Floor	0,1,2,3,4	Floors of each building
Building ID	0,1,2	Each of the three buildings TI,TD, and TC
Space ID	integer	Identification of an indoor space (e.g. room or corridor)
Relative Position	1= inside 2=outside	Outside means in front of the door of the space
User ID	integer	Unique code for users who participated in the experiment
Phone ID	integer	Android device code used for this experiment
Timestamp	Unix time format	Time of capture

happened on these six discrete days. The day when the greatest amount of data was collected was the 20^{th} of June. This single day accounts for more than half the points in the entire data set.

A set of pre-processing tasks were performed to improve the quality of the data set and make the measurements suitable for cluster analysis. The timestamp was recorded in a Linux-based time format, such as 1371714467, which corresponds to `06/20/2013 at 7:47 am`. All the time values were converted to the standard date-time format (dd/mm/yy hh:mm:ss) in Python and added as a new feature called Datetime into the data set.

The feature Space ID contains values starting from 1 to 250, but there were gaps between numbers 50-100 and 150-200. These gaps were removed, and the updated Space ID numbers range from 1 to 122. These space ID values were not unique in terms of spaces such as offices, classrooms, and labs. The Space ID, Building ID,

Floor, and relative position together provide a unique label for a location. A new feature is added to the data set called SPACEID, that contains a unique spatial identifier for each location. The SPACEID was created by combining the space identifiers and then giving them sequential numbers from 0-732.

During the six days of collecting data, some phones (Phone ID = 0,2,4,5,9,12,15,20, and 21) were not used. To make it consecutive, gaps removed and a new feature called PHONEID added to the data with numbers between 1 to 16 for all presented phones. Table 5.5 shows the phone ID related to each android device, the version of the operating system and finally, which user uses the phone. A total of 18 users with sixteen different phones are used for this research. The phone number 9 shared between three users with numbers 1, 9, and 16.

Table 5.5: Correspondence between PhoneID, real device and userID

PhoneID	**Android Device**	**Android Version**	**UserID**
1	GT-I8160	2.3.6	8
2	GT-I9100	4.0.4	5
3	GT-S5360	2.3.6	7
4	GT-S6500	2.3.6	14
5	Galaxy Nexus	4.2.2	10
6	HTC Desire HD	2.3.5	18
7	HTC One	4.1.2	15
8	HTC Wildfire S	2.3.5	11
9	LT22i	4.0.4	1,9,16
10	LT26i	4.0.4	3
11	M1005D	4.0.4	13
12	MT11i	2.3.4	4
13	Nexus 4	4.2.2	6
14	Orange Monte Carlo	2.3.5	17
15	Transformer TF101	4.0.3	2
16	bq Curie	4.1.1	12

For cluster analysis the data collected during six days are considered. The hyperparameters for the AP in the DSAP algorithm are consistent and fixed for the entire experiment. The damping value is equal to 0.96, maximum iteration is 100,

and preference is equal to the median of the similarity matrix.

Finally, the variable selection task was performed to prepare a target data set ready to be used by the two phase clustering algorithm. For this research, two numerical variables are selected as summarised in Table 5.6.

Table 5.6: Selected input variables for clustering the data points

Variable	**Description**	**Values**
PHONEID	Each phone represents a participant	Range between 1 - 16
SPACEID	Each space represents a location	Range between 0 - 732

5.7 System and Software

Various tools and languages were used and they are listed below.

- *Platform:* This research work was performed on a personal computer running on a 64-bit operating system, window 10 Enterprise, 2019. The specifications of this machine are: Intel Core i7 2.60GHz CPU, 32GB internal memory RAM, and 3TB external memory.

- *Language selection:* DSAP is developed in Python. The version of Python used is 3.7. Spyder 3.3.6 IDE which is part of Anaconda 2019 was used for project development. Main libraries applied in Python are sklearn, pandas, numpy, matplotlib, datetime, scipy, psutil, itertools and skmultiflow.

- *Software:* Microsoft Excel 2020 was used for preliminary visualization of the .CSV file format. It has a neat feature to quickly filter the data and visualize it.

  Lucidchart web-based platform is another software that was used for drawing, revising and sharing charts and diagrams.

Chapter 6

Discussion of the Results

In this Chapter, the main clustering results from applying stream clustering using the data streams generated from both experiments are presented to discuss the strengths and weaknesses of the proposed DSAP algorithm.

6.1 Behavioural Intervention Experiment

The streaming cluster analysis was aimed at finding any evolutionary patterns in stair usage due to an educational intervention campaign. The clustering results were generated using three months of the experiment, which were named as before, during, and after the intervention.

6.1.1 Spatio-temporal Data Patterns

The most relevant measurements for this cluster analysis were the location of people counter sensors at the Tonsley building and the number of people passing through these sensors. The landmark time window of one hour was used to accumulate the data streams. Figure 6.1 illustrates the daily stair usage patterns. The three months named as before intervention, during the intervention, and after intervention are highlighted in purple, blue, and yellow shaded regions, respectively. It is quite

apparent that more people preferred taking the stairs to go down rather than up. Expected low usage of stairs is also observed on the weekends. The mid-semester break from April 15-28 was excluded from the analysis as the measurements made during this period represented anomalous behavior.

To gain further insights into the data, Figure 6.2 shows the number of people using the stairs (both up and down) for each month of the experiment. No discernible differences could be found in these patterns that were observed between the weekdays for one month of data collected before, after, and during the intervention. The weekends as expected, reported count values orders of magnitude smaller than the weekdays. Although the overall trend of the number of people using the stairs at each level remains consistent for the three months, one peak of stair usage occurred on Monday, April 1, before the intervention, another peak occurred on Friday May 10, during the intervention and happened again on Tuesday, June 11 after the intervention campaign has ended. After a closer inspection of the data, it was found that on May 10, a group of students went up around 12:06 pm, and presumably, the same group came down at 12:23 pm. This led to an increase in the measured count across many levels, whether this alone accounts for the increase in the accumulated count values for the entire month after the intervention. No particular event was found to justify other peaks.

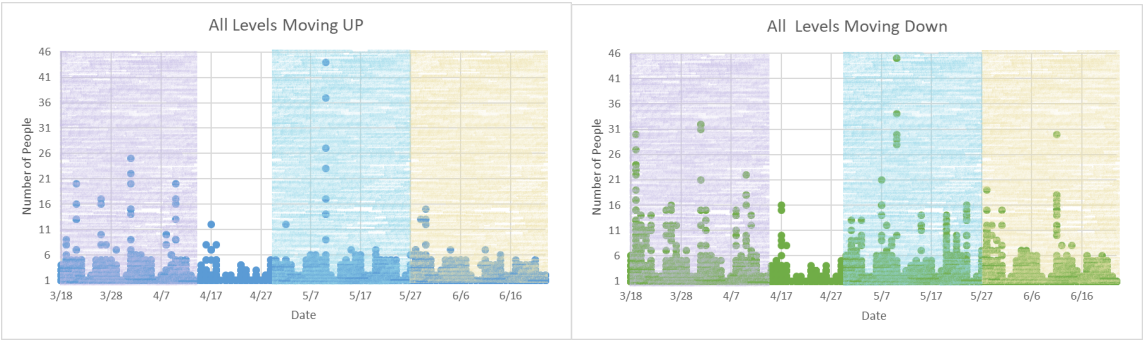


Figure 6.1: Total number of people using the stairs: before intervention month (purple), during intervention month (blue), and after intervention month (yellow)

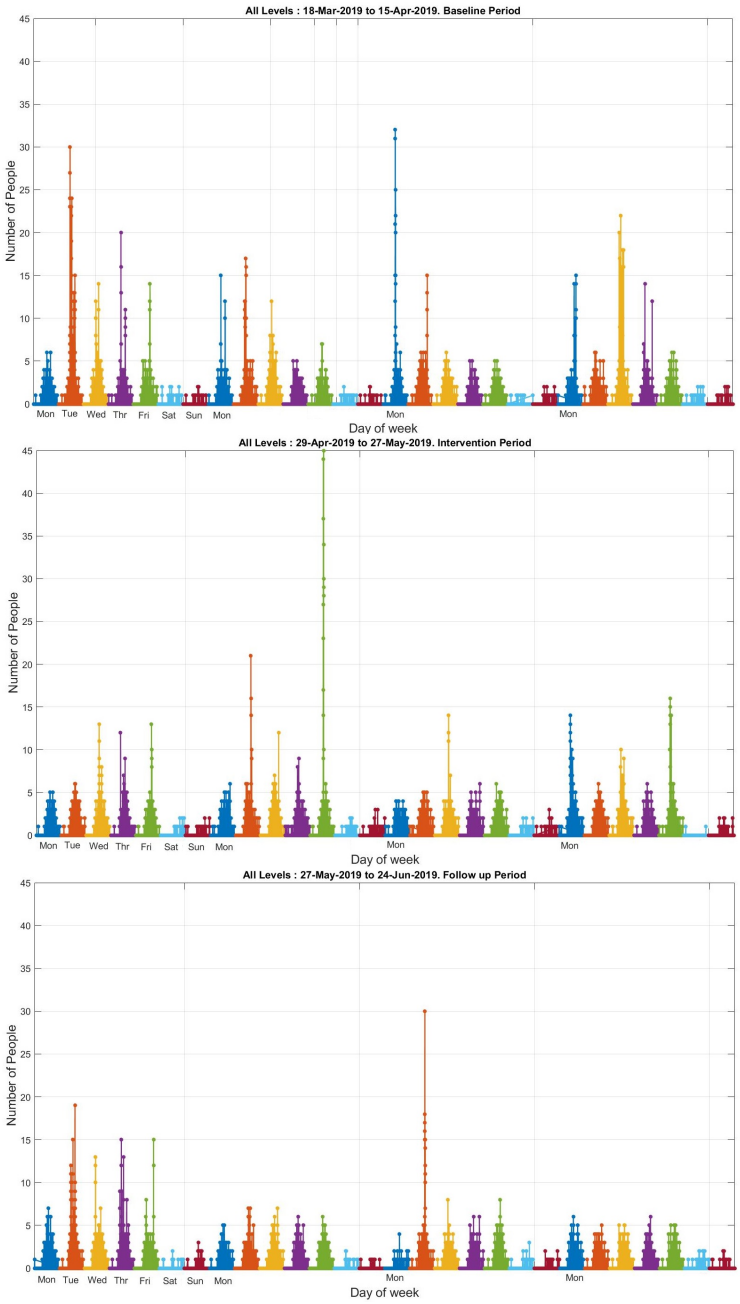


Figure 6.2: Weekly patterns of stair usage before, after and during intervention: Sunday (light blue), Monday (red), Tuesday (dark blue), Wednesday (orange), Thursday (yellow), Friday (purple), Saturday (green)

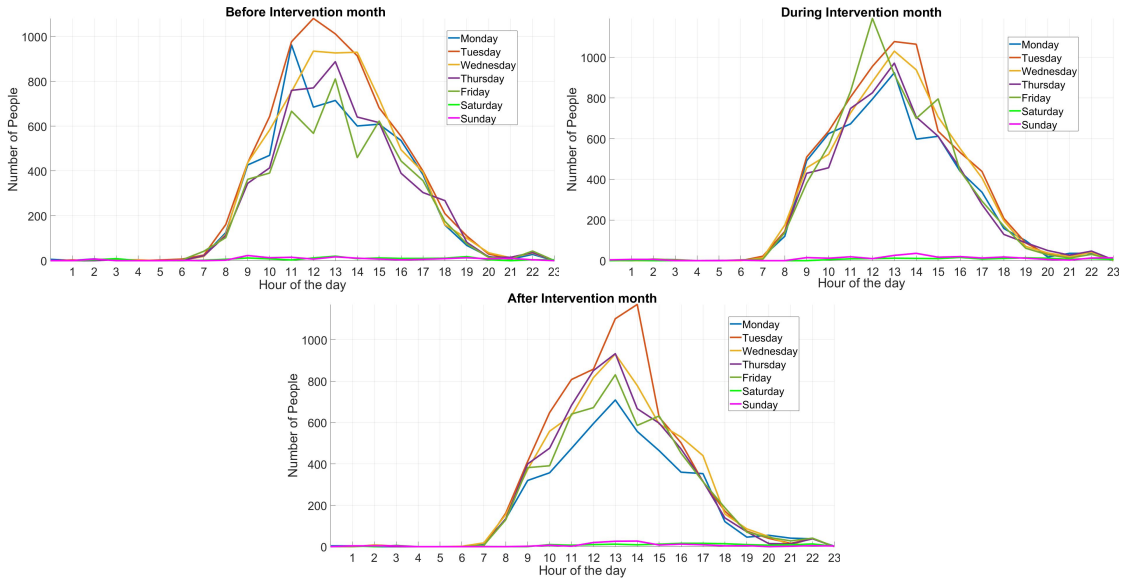


Figure 6.3: Hourly stair usage patterns for each month: before (top), during (middle), and after (bottom) intervention

A dynamic pattern behaviour was also observed on an hourly basis for each month of the experiment, as shown in Figure 6.3. The usage peak is around noon for most of the weekdays, most probably due to the lunch break. The people count is asymmetric on both sides of the peak, with more people using the stairs on Tuesday afternoons during all months of the experiment. Overall the month before the intervention campaign seems to have recorded the highest usage than the following months. By looking at this trend, our preliminary hypothesis is that the intervention campaign has not generated the expected impact on motivating people to take the stairs. Therefore, the clustering results play a significant role in testing this hypothesis.

The spatio-temporal patterns found in Figure 6.4 were generated by aggregating the number of people at each level of the building using the three months under analysis. It is clear from the graphs on the left of Figure 6.4 that the Level 2-1 stair were frequently used; meanwhile, levels 4-3 North and 5-4 North were used the least. This could be due to fewer people using the north entrances since the north entrance leads to a low capacity parking lot (refer to Figure 5.2 for the Tonsley building

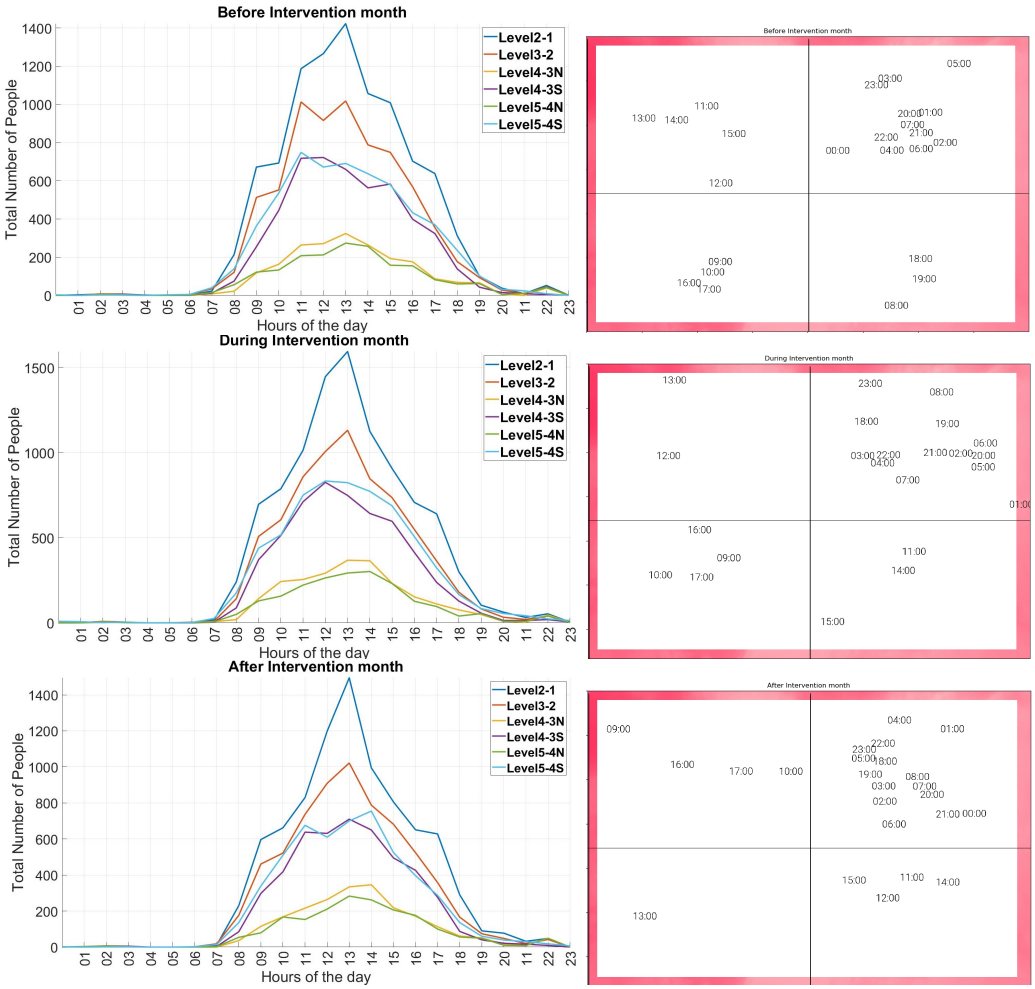


Figure 6.4: Spatio-temporal patterns during the experiment. *Left*: Hourly stair usage patterns per building level. *Right*: Similar bins of hourly cluster patterns

layout). These aggregated plots also confirm that the usage peak was around noon. The asymmetry in people count on both sides of the peak is seen in these plots as well. This signifies that consistently people have used the stairs in the afternoon rather than in the morning.

Finally, Figure 6.4 shows the clustering results that were generated by applying the K-means clustering algorithm on the total number of people found at each hour across all levels. Four bins are used to distinguish the hourly cluster results. The usage peak occurred between 12 pm and 1 pm at Level 2-1 for both during and after intervention months. The stair usage at 11 am, 2 pm, and 3 pm. shows the

highest values. During the late evening and early morning hours is expected that the number of people inside the building was low. The last bin consists of the hours with moderate activity seen at the start and end of the workdays.

6.1.2 Micro-cluster Evolution

The evolution of micro-clusters was interesting to be analyzed in order to detect any concept drift during the clustering process. In this Section, we discuss the results generated on Wednesday, May 1, which were observed between 7 am to 7 pm. Figure 6.5 shows the centroids of the micro-clusters and their respective data points that are connected using straight lines. The one-hour expiration time was chosen to ensure that only the data points within the one-hour window were used for finding clusters.

The changes in stair usage across all levels can be clearly seen in Figure 6.5. The micro-clusters show an increase in activity starting around 9-11 am. At the peak around noon, the micro-clusters become more dense occurring at different levels of the building. As the day progresses, the number of micro-clusters gradually decreases. These generated patterns were observed throughout the experiment, demonstrating the potential of using the DSAP approach for finding and tracking meaningful micro-clusters over space and time. The strength of DSAP lies in its ability to continuously find micro-clusters by responding quickly to any changes in the stream data.

6.1.3 Generated Micro and Macro-clusters

The DSAP clustering results are presented by visualizing all the centroids of the micro-clusters as blue circles while the centroids of the macro-clusters are represented as red crosses. The data points are visualized as grey dots. The numbers of data points associated with each micro-cluster are mentioned right next to it. Searching for meaningful clusters has been achieved by evaluating weekly and monthly patterns.

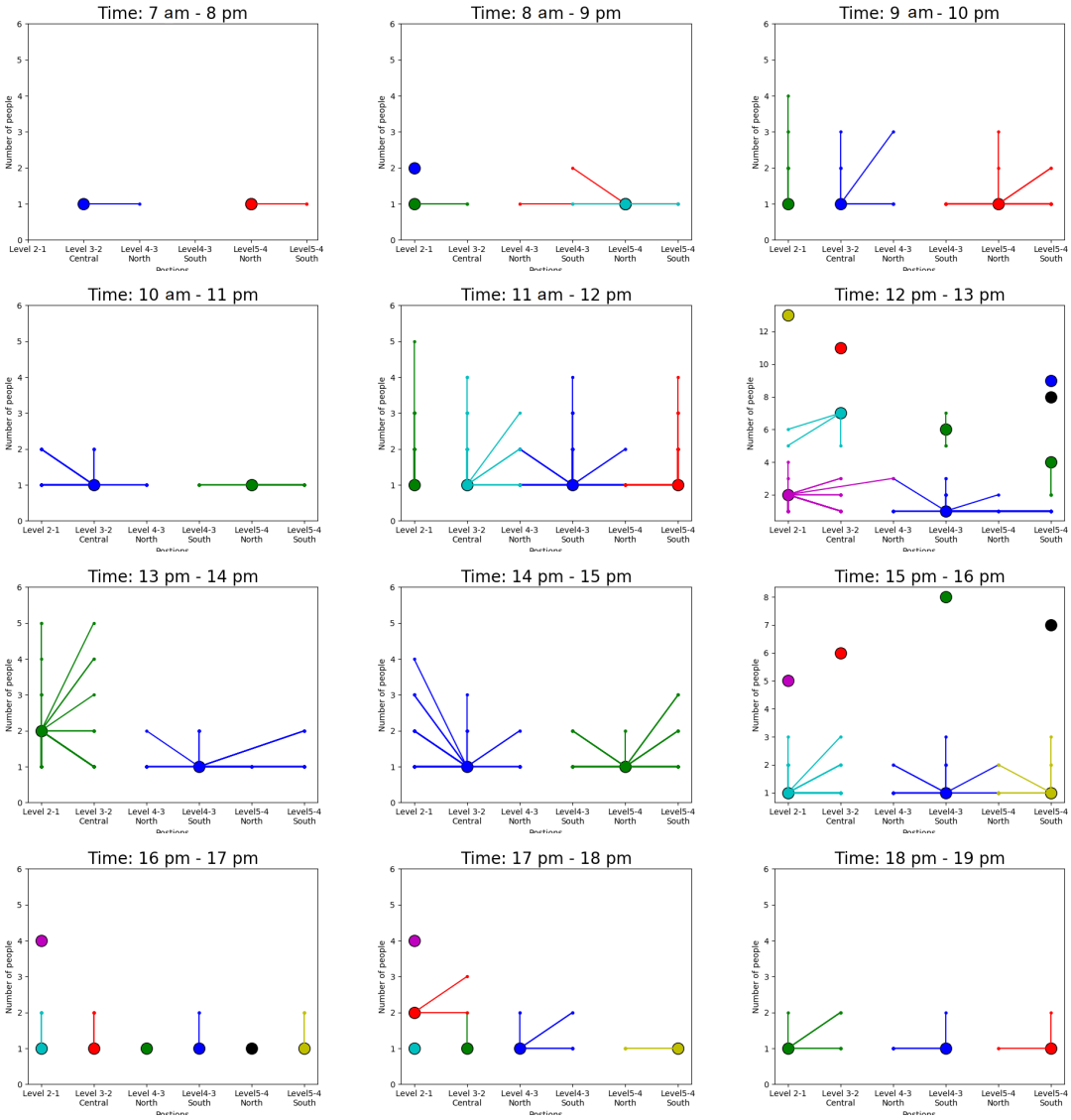


Figure 6.5: The evolution of micro-clusters between 7 am to 7 pm on Wednesday May 1, 2019

Table 6.1: Weekly Clustering Statistics

Macro-Cluster	Coordinates in the Plot	Number of Micro-Cluster	Total Number of Data Points
First Week Before Intervention (March 18 - March 24)			
1	(Level 2-1, 1)	1	1603
2	(Level 2-1, 9)	3	64
3	(Level 4-3S, 1)	6	4520
4	(Level 4-3S, 22)	2	13
5	(Level 5-4S, 5)	3	82
First Week During Intervention (April 29 - May 5)			
1	(Level 2-1, 5)	3	109
2	(Level 3-2, 1)	5	4388
3	(Level 3-2, 11)	3	6
4	(Level 4-3S, 1)	1	921
5	(Level 5-4S, 1)	3	1664
First Week After Intervention (May 27 - June 2)			
1	(Level 2-1, 3)	4	3269
2	(Level 3-2, 15)	3	11
3	(Level 4-3S, 1)	1	1810
4	(Level 4-3S, 8)	4	36
5	(Level 5-4S, 3)	4	1207

Weekly Patterns

The first week of each of the three months under study was selected to compare the clustering results. The first week of March 18 - 24 before the intervention, the first week of April 29 - May 5 during the intervention, and the first week after the removal of the educational aids was from May 27 - June 2. The expiration time equal to 168 which is equal to total number of hours per weeks, was used for generating weekly patterns that are summarized in Table 6.1. Five macro-clusters have been found for each week, having a variety of density values ranging from 6 up to 4520. This overall trend of high-density clusters with low counts was consistently present in data. It is also interesting to point out that the same macro-cluster (level4-3S,1) has happened during the three weeks of the analysis, but having a different number of micro-clusters containing a large number of data points.

Figure 6.6 illustrate the occurrence of the weekly patterns at different levels

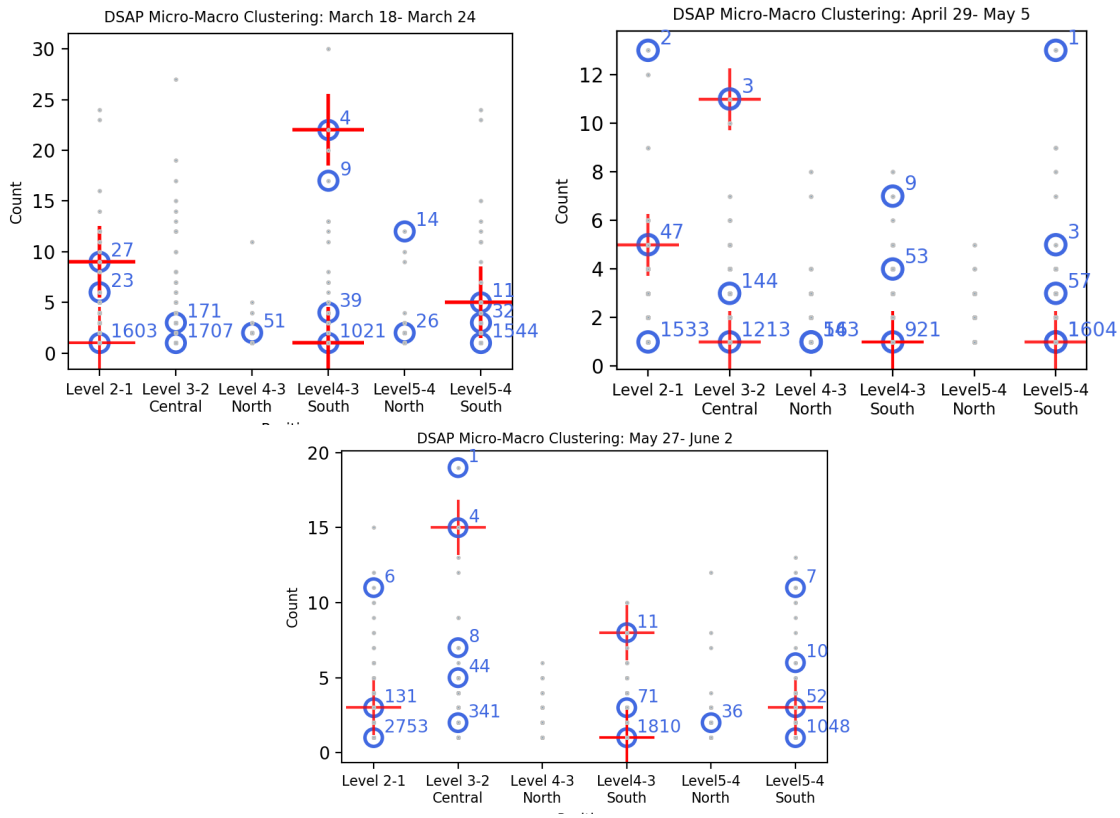


Figure 6.6: Clustering results for the first week before, during, and after the intervention campaign

of the building, revealing that the intervention campaign had a positive impact on people taking the stairs between level 3-2 and level 4-3 south. In contrast, the campaign has also shown to have a weak impact in changing the behaviour of people when taking the stairs between level 4-3 north and level 5-4 north. Finally, the intervention campaign has not increased the usage of stairs between level 2-1. No anomalous clusters were found in these results.

Monthly Patterns

The three months of the experiment have been chosen to see the final pattern out of clustering results. The expiration time value of 670 was used to compute the monthly patterns as shown in Table 6.2. For the first month, eight macro-clusters, during the intervention, twelve and after intervention ten macro-clusters have been

Table 6.2: Monthly Clustering Statistics

Macro-Clusters	Coordinates in the Plot	Number of Micro-Clusters	Total Number Of Data Points
First Week Before Intervention (March 18 - April 14)			
1	(Level 2-1, 1)	1	6251
2	(Level 2-1, 6)	1	88
3	(Level 3-2, 1)	1	6471
4	(Level 3-2, 31)	2	3
5	(Level 4-3S, 12)	1	53
6	(Level 4-3S, 17)	1	22
7	(Level 5-4S, 3)	2	6922
8	(Level 5-4S, 23)	2	16
First Week During Intervention (April 29 - May 26)			
1	(Level 2-1, 5)	1	183
2	(Level 2-1, 13)	1	8
3	(Level 2-1, 23)	1	1
4	(Level 2-1, 30)	1	2
5	(Level 3-2, 3)	2	13460
6	(Level 3-2, 11)	1	16
7	(Level 3-2, 16)	1	5
8	(Level 4-3S, 2)	3	4068
9	(Level 4-3S, 7)	1	24
10	(Level 5-4S, 5)	2	816
11	(Level 5-4S, 13)	1	9
12	(Level 5-4S, 22)	1	1
First Week After Intervention (May 27 - June 23)			
1	(Level 2-1, 1)	2	10846
2	(Level 2-1, 11)	1	8
3	(Level 3-2, 7)	3	144
4	(Level 3-2, 15)	1	8
5	(Level 3-2, 19)	1	3
6	(Level 4-3S, 1)	1	6819
7	(Level 5-4S, 2)	4	4188
8	(Level 5-4S, 6)	1	19
9	(Level 5-4S, 11)	1	8
10	(Level 5-4S, 18)	1	4

found. The total number of data points associated with micro-clusters are varied between 1 up to 13460. Interestingly, all macro-clusters for these three months are at levels 2-1, 3-2, 4-3 south and 5-4 south, confirming that these levels saw the highest frequency of people.

The location pattern of very dense clusters for the month before and after intervention are similar. All three periods have three dense clusters each; Level 2-1, 3-2, and 5-4 south are the cluster centers representing around 6000 data points for the month before intervention. Interestingly two moderately dense clusters with the number of data points equal to 53 and 22 respectively were observed at Level 4-3 South with count values of 12 and 17. These clusters most probably correspond to a group of people using stairs during this month. This behavior is not seen in the other two months.

The month after the intervention has three dominant clusters at Level 2-1, 4-3 South, and 5-4 South representing 4000-10000 data points. These clusters represent levels with the most amount of data points. No anomalous behavior is noticed during this month, unlike the month before intervention. However, for the month of intervention, the pattern is slightly different. The densest cluster at Level 3-2 represents 13460 data points which is almost three times as dense as the other two prominent clusters at Level 4-3 South and Level 5-4 South.

Figure 6.7 depicted the monthly pattern generated by the DSAP clustering algorithm at different levels of the building during the three months of the experiment. It is clear that with the same values of hyperparameters, during intervention gave the most number of macro-clusters. The more dense clusters are clearly found at the lower levels. Level 5-4 south is the only stair that keeps its behavioural pattern similar for the entire three months.

The people counter experiment was performed in order to find any improvement in stair usage after some measures for educational interference was made. In

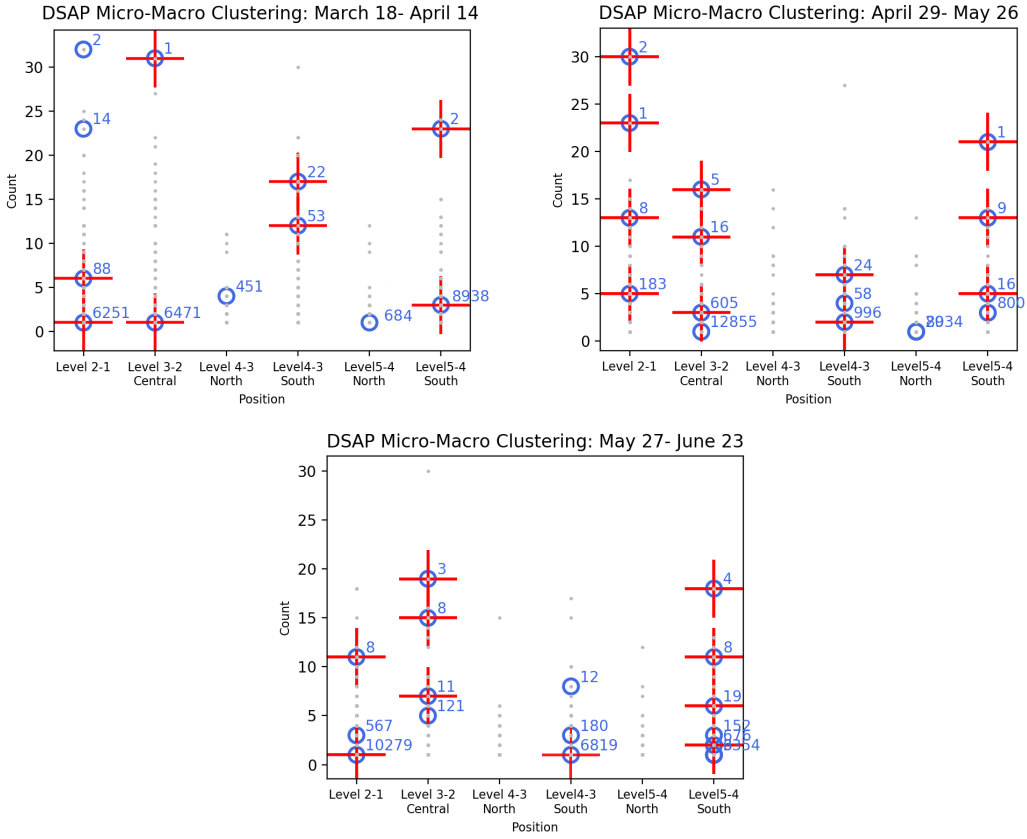


Figure 6.7: The clusters obtained by the DSAP algorithm from the e-counter data for one month periods before, during, and after intervention

order to analyze the data, the DSAP algorithm was applied on each of the months before, during, and after intervention using a landmark time window of 1 hour. The clustering results find regular behavior during each of the three months. One potentially anomalous cluster is seen during the month before intervention. There are no obvious signs for an overall increase in stair usage during and after the month of intervention.

6.1.4 Performance and Validation Metrics

For the DSAP evaluation, the AP model was used as a baseline to compare the performance and validation metrics. We have selected AP because of two main reasons. First, Other AP stream clustering algorithms and their codes were not

available at that time. Other streaming algorithms such as streaming K-means in the R programming language use different clustering methods, in this case, random sampling. Moreover, the DSAP algorithm is based on AP, and we expected to gain similar results.

The same weeks used for computing the macro-clusters shown in Figure 6.6 were used in this evaluation. The results for each of the metrics, along with the number of clusters, processing time, and memory consumption, are summarized in Table 6.3. A significant saving in processing time is immediately apparent between the DSAP algorithm and AP. The processing time here includes the time taken for all the phases and steps of the DSAP algorithm as it analyzes the entire data using time windows. For a real stream, the execution time of interest would be the time taken to execute one window and is expected to be much lower as in DSAP. The initial window calculation is the most time-intensive step but is performed only once and does not affect the current time window.

The accuracy of the two methods is similar, but the micro-clusters generated by DSAP show slightly improved values, as shown by the validation metric values. The number of micro-clusters generated by DSAP are similar to clusters generated by AP suggesting that the activity levels remain consistent. The intrinsic validation metrics, Silhouette and Davies-Bouldin indices for the micro-clusters have similar values to the numbers for AP clusters. The number of micro-clusters is the same order as the macro-clusters, hence, some of the validation metrics are not accurate. It should be noted that the number of points represented by a cluster significantly affects some of these metrics, and hence internal evaluation metric numbers need to be carefully interpreted.

Table 6.3: Overall results from the performance evaluation and clustering validation

Metrics	AP Algorithm	DSAP Algorithm
Before Intervention Week (March 18 - March 24)		
Processing Time (s)	22	0.6
Memory Consumption (MB)	536	250
Number of Clusters	14	micro = 15 macro = 5
Silhouette Index	0.4	micro = 0.7 macro = 0.4
Caliński-Harabasz Index	2009	micro = 4872 macro = 87
Davies-Bouldin Index	0.5	micro = 0.8 macro = 0.5
Intervention Week (April 29 - May 5)		
Processing Time (s)	25	0.6
Memory Consumption (MB)	536	256
Number of Clusters	11	micro = 16 macro = 5
Silhouette Index	0.5	micro = 0.5 macro = 0.4
Caliński-Harabasz Index	2421	micro = 6292 macro = 35
Davies-Bouldin Index	0.7	micro = 1 macro = 0.2
After Intervention Week (May 27 - June 2)		
Processing Time (s)	26	0.6
Memory Consumption (MB)	541	260
Number of Clusters	12	micro = 16 macro = 5
Silhouette Index	0.4	micro = 0.6 macro = 0.4
Caliński-Harabasz Index	1278	micro = 4381 macro = 40
Davies-Bouldin Index	0.5	micro = 0.5 macro = 0.5

6.2 Occupant Behaviour Experiment

The streaming cluster analysis was carried out to reveal occupancy patterns in different buildings at the Universitat Jaume I campus, in Valencia, Spain. The clustering results were generated from analyzing data streams generated by 16 participants during one day of the experiment. The experiment lasted for six days from May 30 to June 20, but June 20 was selected for the streaming cluster analysis since significantly more data points were collected on this day as shown in Figure 6.8. The first five days from May 30 until June 12, data streams were only available for building T1, but on June 20, data streams were available for both buildings TD and TC. Finally, the only day that almost all users were participating in the experiment was June 20.

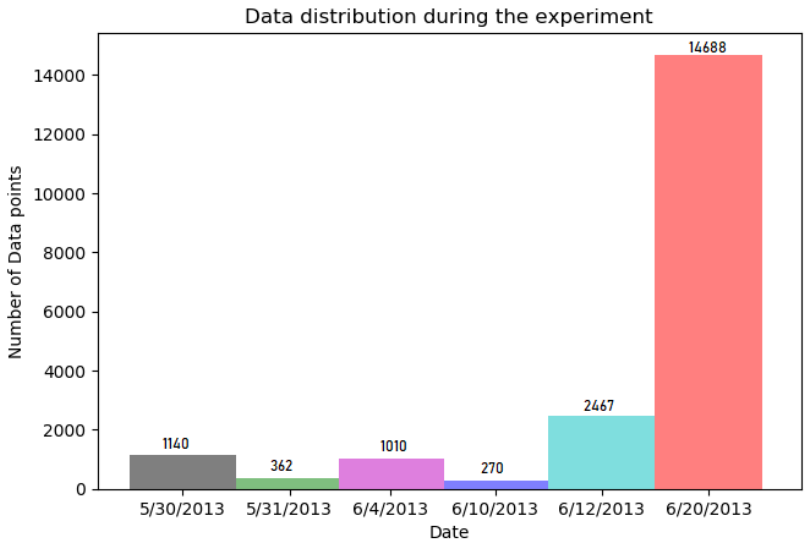


Figure 6.8: Number of data points generated from May 30 to June 20, 2013

6.2.1 Spatio-temporal Data Patterns

The most relevant observations for this study were the location of the user (SPACEID) and user's phone (PHONEID). A 10-minute landmark time window was

chosen to analyze the UJIIndoorLoc data. The assumption was that the time interval of 10 minutes would accumulate a sufficient number of data points for computing the micro-clusters without having issues with overflowing the time-based repository. A large number of data points were generated every minute during this experiment.

Figure 6.9 shows an overview of the spatial distribution of all data points according to the indoor spaces of the three buildings named as T1, TD, and TC, including their four floors except the TC building that has the 5th floor.

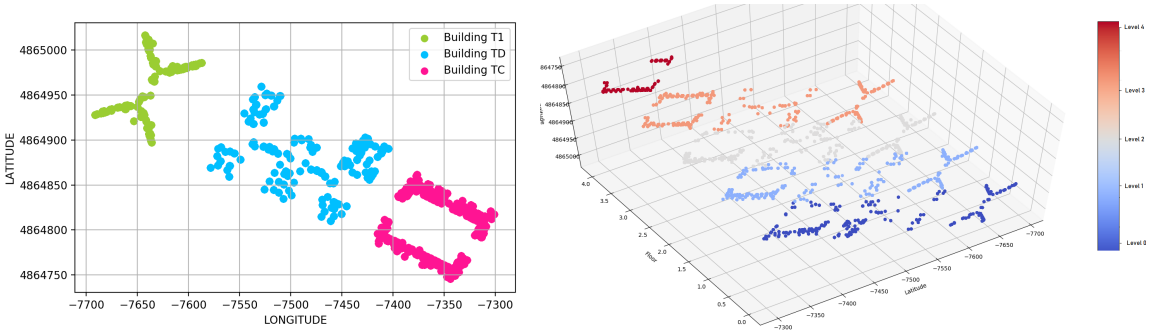


Figure 6.9: 2D and and 3D views of the location of all data points generated during the experiment

In Table 6.4 the total number of data points generated for each each building is provided. In fact, the TC building has contributed to 47% of the whole data. This is due to the additional measurements made in the building on the 5th floor. Moreover, the total number of data points gathered at different floors of all buildings was very similar, (i.e. just over 20%), with the exception of the 5th floor of the TC building (i.e. 5.7%) as shown in Figure 6.10.

Table 6.4: Total number of data points generated per building

Building	Number of Data Points
T1	5249
TD	5196
TC	9492

For the stream cluster analysis, the space identifier was used as a proxy to

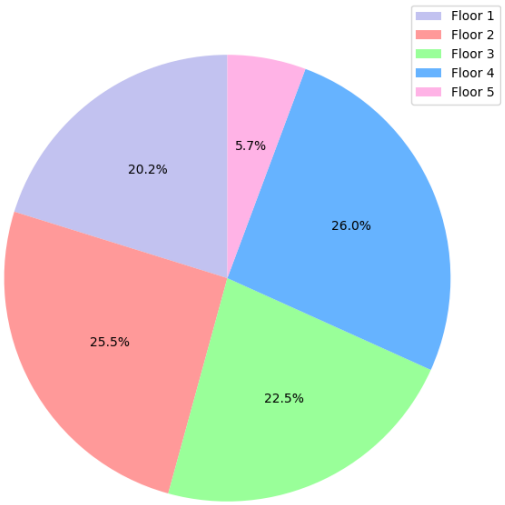


Figure 6.10: Percentage of data points generated for each floor of all buildings

represent the indoor space that a participant was occupying during the experiment. This SPACEID provides a unique identification for indoor spaces such as rooms, classes, lab, and offices. It has a number ranging from 0 to 732 for buildings T1, TD, and TC. SPACEID from 0 to 255 belongs to building T1, numbers between 256 to 416 belong to building TD, and the remaining numbers (417 to 732) belong to building TC. The total number of data points generated at each indoor space is illustrated in Figure 6.11.

The analysis was carried out using the data streams generated by 18 participants' phone. Figure 6.12 illustrates the overall distribution of the total data points collected by individual phones. The participants with the phone number 8 and 9 generated most of the data during the experiment.

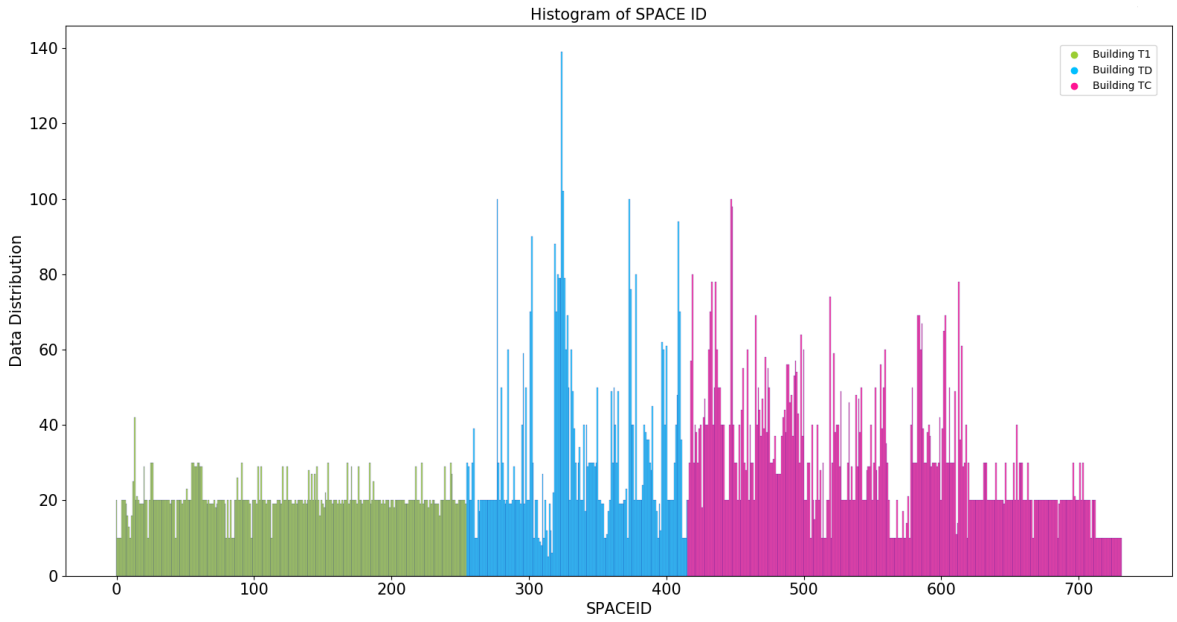


Figure 6.11: Total number of data points generated in indoor spaces of the T1 (green), TD (blue), and TC (pink) buildings

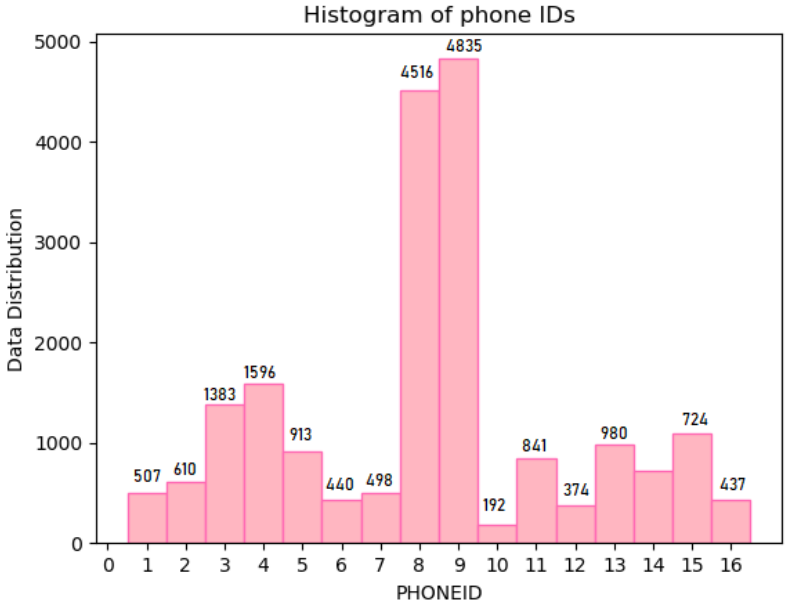


Figure 6.12: Data distribution of the total number of data points generated by the participants of the experiment

6.2.2 Micro-cluster Evolution

As a part of this experiment, the limited number of participants have moved across several indoor spaces; hence it is important that the algorithm is able to find

representative micro-clusters that reflect the changes in the occupants behaviour. This evolutionary process can be seen in Figure 6.13 happen on June 20, between 9:15 am to 10:35 am, with a 10-minute time window and an expiration time of one window. The patterns show that two or more users rarely occupy the same indoor space. They also show how the micro-clusters provide an overview of the movement in the buildings. Therefore, the strength of the DSAP model is the ability to detect and track people's occupancy over a short period of time, in this example 90 minutes.

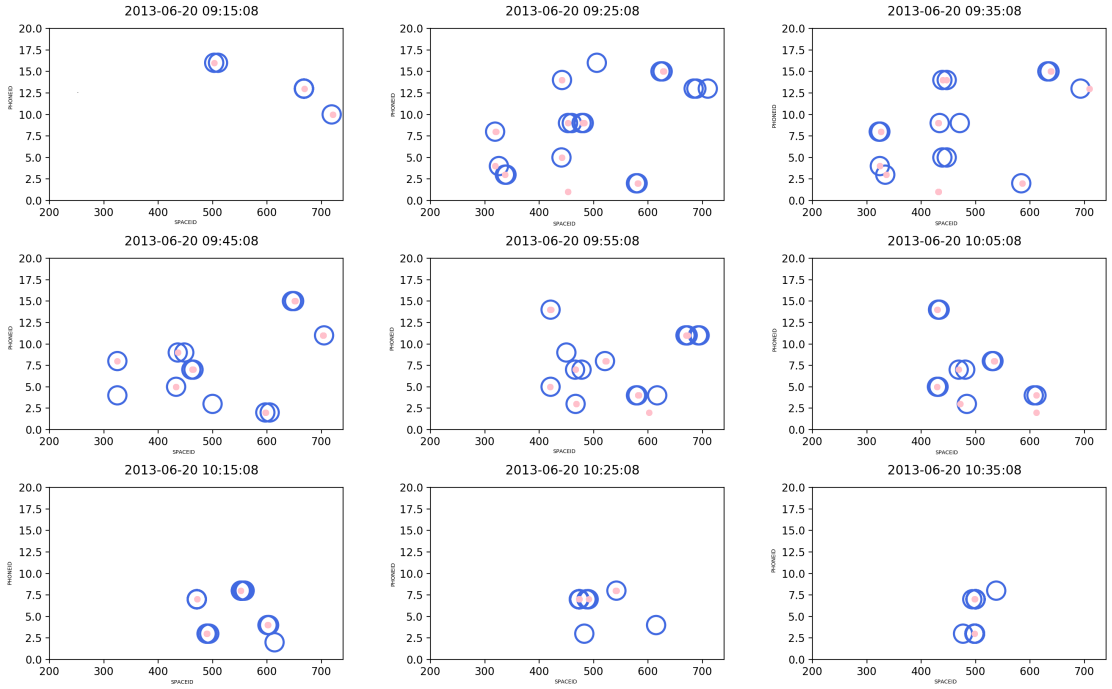


Figure 6.13: The evolution of micro-cluster. The blue circle are centroids and red dots are data points with the time frame of every 10 minutes.

6.2.3 Generated Micro and Macro-clusters

The DSAP clustering results for the day June 20 are presented in Figure 6.14. In the first plot, micro-clusters are represented by blue circles and the macro-clusters are visualized by red crosses. The data points are depicted as grey dots. The numbers of data points associated with each micro-cluster are mentioned right next to it. The second plot is a summary of this day. DSAP successfully found macro-clusters

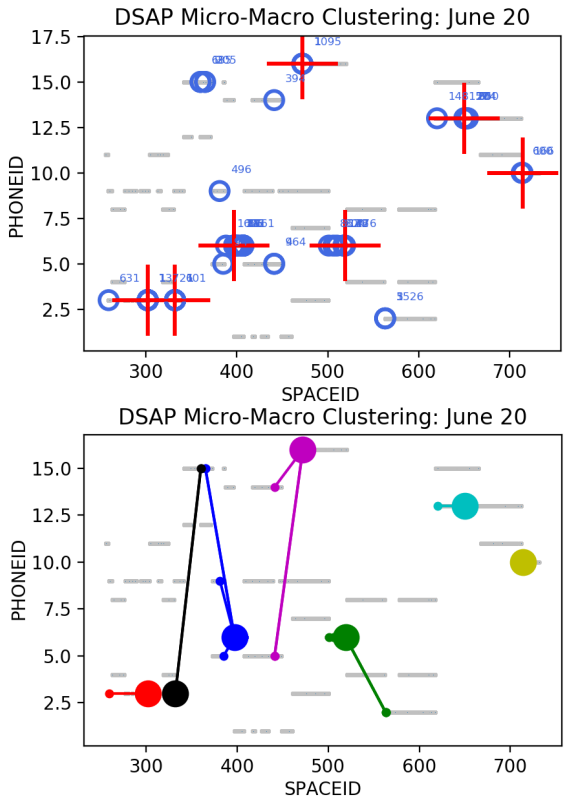


Figure 6.14: DSAP for UJIIndoorLoc data during the experiment: June 20. The first plot shows the micro and macro-clusters, and the second plot shows the macro-clusters with corresponding micro-clusters.

centered in the indoor spaces 400, 500, and 659, but in addition, it has also found another set of clusters in SPACEID 300. It should be noted that the x and y axes are not of the same scale; hence even though the branches of the clusters might seem like that they are far apart, but they, in fact, belong to similar indoor spaces.

In rapidly changing data like this dataset, the choice of the expiration time is critical in order to interpret the data meaningfully. Figure 6.15 is the DSAP algorithm clustering results for the entire experiment duration of six days across the three buildings with DSAP expiration values of 20000, 2000, 1000, and 200. As mentioned before, the first five days of data were collected from the first building (T1) with the SPACEID range of 0 to 255. The first figure (top left plot) shows DSAP clusters for almost the entire data using an expiration value of 20,000. By decreasing the value of expiration to 2000, the occupancy behaviour changes from

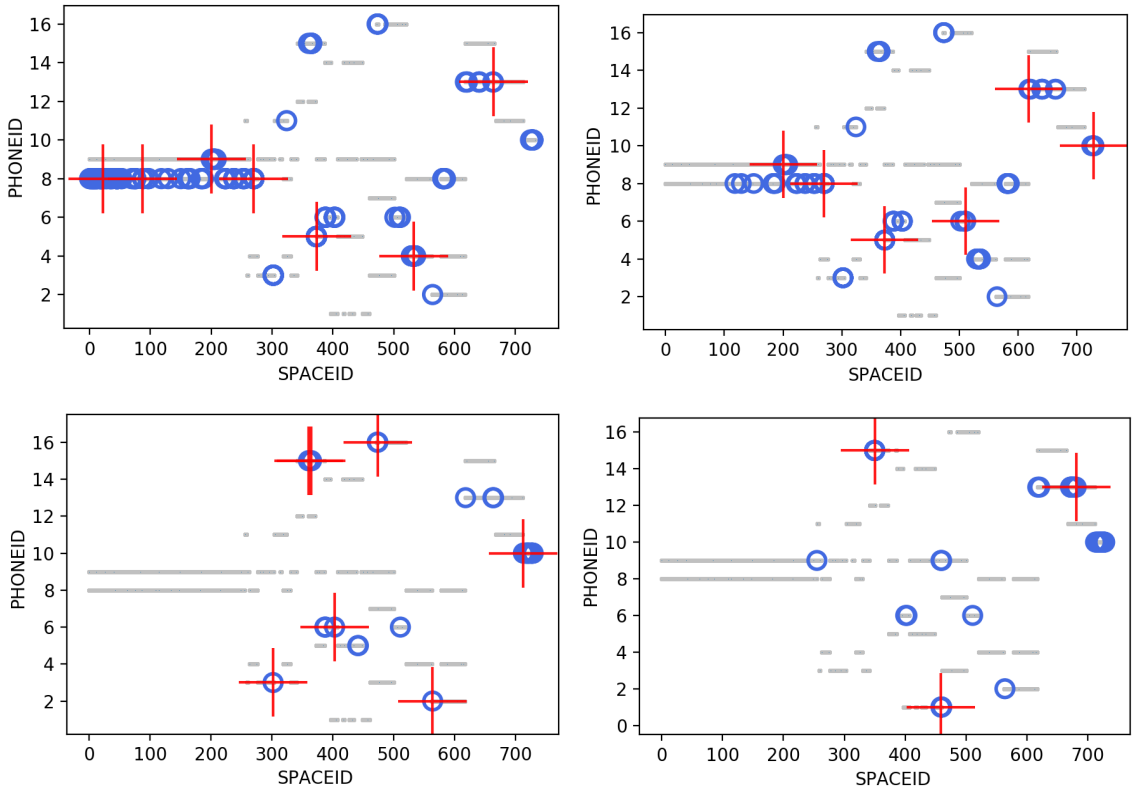


Figure 6.15: Four different expiration time for the total number of data points. These numbers are 20000, 2000, 1000, and 200 respectively.

the first 200 SPACEIDs related to building T1 to the subsequent spaces and different participants, as seen in the top right plot.

The bottom left plot in Figure 6.15 shows the DSAP results with an expiration value equal to 1000. These last 1000 windows no longer retain any information from older data measured at building T1 but instead represent occupancy patterns in building TD and TC, mostly with the SPACEID between 300 to 700. The last figure on the bottom right has fewer macro-clusters as compared to the previous one due to the lower number of micro-clusters and changes detected by DSAP. These plots show how the DSAP can detect the changes in data over time and how well windows can update the results and summary of clusters.

6.2.4 Performance and Validation Metrics

Once again, the AP model was applied as a baseline to compare the performance and validation metrics. The clustering results found on June 20 were selected for this evaluation. The DSAP and AP results of each metric, along with the number of clusters, processing time, and memory consumption, are summarized in Table 6.5. The DSAP results significantly improve the processing time and memory consumption. The processing time includes running the DSAP during online and offline phases and plotting the final results. The processing time for this data is higher than the e-counter data and it is because of the number of data points is almost three times compare to that data set. Also, the number of micro and macro clusters affect the processing time and memory consumption.

Table 6.5: Overall results from the performance evaluation and clustering validation

Metrics	AP Algorithm	DSAP Algorithm
June 20, 2013		
Processing Time (s)	414	4
Memory Consumption (MB)	1933	291
Number of Clusters	5	micro = 101 macro = 7
Silhouette Index	0.3	micro = 0.8 macro = 0.3
Caliński-Harabasz Index	7193	micro = 19512 macro = 1206
Davies-Bouldin Index	0.7	micro = 1 macro = 0.2

The memory consumption is very low, which solves AP's problem of handling very large data. The metrics of assessing the quality of clustering for both algorithms show that both AP and DSAP have similar clusters. Owing to the dynamic nature of this data, the number of micro-clusters generated by DSAP are a lot higher. The number of macro-clusters represents the overall pattern and should be compared with the AP results.

Chapter 7

Conclusions and Future Work

7.1 Summary

With the increasing number of IoT devices, a staggering amount of data streams are being generated in a continuous flow. These data streams consist of a sequence of data points that need to be analyzed using processing techniques that do not have access to all data. In this research work, the DSAP model was developed for uncovering evolutionary patterns based on two phases: online micro and offline macro phases. The landmark time window model was proposed to ensure that only the latest data points in the stream are used in the clustering process.

The concept of having expiring data points within a time-window repository was introduced in this research work. The expiration time is a hyperparameter that can be changed based on the different needs of the users. Moreover, the latest landmark time window keeps the micro-cluster centroids of the model updated and the size of the micro-clusters under control. The combination of the time windows and the expiration time enables the DSAP model to handle concept drift and changing data distribution over time.

In the online phase, the arriving data points were analyzed in near real-time and

relevant summary statistics were provided. The results of the online phase consisted of a number of micro-clusters that represented a large number of preliminary clusters in the stream. In the offline phase, the micro-clusters were re-clustered to generate a final set of macro-clusters. The number of macro-clusters was always much smaller than the number of micro-clusters.

The clusters found using the proposed DSAP clustering algorithm were evaluated by applying intrinsic validation indices for unlabeled data such as silhouette index, Caliński-Harabasz index, and Davies-Bouldin index. These metrics were calculated for both the micro and macro-clusters by estimating the centroids for each phase and data points related to each of the centroids. In addition, the performance of the algorithm was evaluated by using the time and space complexity metrics. These metrics have proven that the implementation of the DSAP algorithm is a robust stream clustering approach capable of handling data streams generated by indoor localization systems commonly used in IoT.

The existing AP-based stream clustering models have not been applied for finding clusters from indoor localization data streams prior to this work. Two distinct localization data streams were analyzed using the DSAP algorithm. Preliminary results demonstrate the strength of the algorithm by finding meaningful patterns and providing new insights into the indoor localization data based on the number of people and their locations in an indoor space.

The first data set analyzed was the e-counter data from an experiment performed at Flinders University to capture the impact of a motivational intervention on the physical activity of people. The feature space consisted of two attributes: the number of people passing by a sensor and the position of each sensor at different stairs in the building. Hourly and monthly stair usage patterns were successfully captured using the landmark time window of one-hour intervals. The clustering results showed the behaviour of people during the three months, i.e., before, during, and after the

intervention. Some positive changes in behavioural patterns were observed during some days of the experiment and at specific levels of the building. Therefore, it can be concluded that the intervention campaign had a weak positive impact on people.

The second implementation of DSAP was carried out using the WiFi indoor localization data streams in order to find out the occupancy behaviour in three buildings at the Universitat Jaume I campus. The features chosen for the DSAP algorithm were SPACEIDs that represent the occupancy of a particular classroom, lab, or office, and the PHONEIDs that represent the participants. The algorithm was robust in capturing the occupancy behaviour of the users using a landmark time window of ten minutes.

The overall clustering results were compared with the standard AP clustering algorithm. DSAP performed better than the standard AP model since a major improvement in the processing time was achieved. DSAP was able to reduce the processing time by a couple of orders of magnitude as compared with the AP algorithm. The memory and processing time also demonstrated that DSAP can easily handle continuous data streams.

The metrics of assessing the quality of DSAP clusters such as Silhouette, Caliński-Harabasz, and DaviesBouldin indexes show that the accuracy of micro-clusters are improved, while the same results did not observe for macro-clusters and it is due to the spread clusters.

The main advantage of the DSAP model relies on its simple and straightforward approach but still retaining the strengths of other streaming AP-based algorithms (i.e., StrAP, IStrAP, ISTRAP, and APdenstream) while removing some of the complexities introduced by them. The combination of a user-defined expiration time, a dynamic threshold ϵ_{W_j}, the landmark time window model, and a time-window repository have been crucial to developing a manageable, fast and accurate clustering algorithm. DSAP runs entirely autonomously without the need to specify the

number of clusters once the hyperparameters are selected.

Due to stream data availability, the DSAP model was not implemented using live streaming data. Therefore, the main limitation of the DSAP model is that it does not handle latency and bandwidth problems, which can occur very often when analyzing live streaming data. Stream data clustering of multiple variables is also expected to bring scaling issues. Highly dynamic data with large concept drift is expected to reduce the performance of the DSAP algorithm as new clusters will be formed in almost every window slowing down the entire process. These limitations are planned to be addressed in future versions of the algorithm.

7.2 Future Work

The DSAP model opens up exciting new avenues for future work not only on real-time data analytics across various sectors but also in model development and implementation. The code is freely available, and its modular framework enables the easy addition of new features. For example, another distance function such as Manhattan distance can be easily used to compute the threshold ϵ_{W_j} without changing the DSAP source code. The shape of the clusters is influenced by choice of the distance function. Multiple distance functions can be used to find the optimal function depending on the data distribution and shape of the clusters. However, changing the landmark time window model to another time window model needs further research to understand the impact in finding macro-clusters. It is crucial to carefully select the time window model taking into account the type of data streams of an application.

The performance of the DSAP model needs to be tested using real-world data streams. Future research work will focus on improving the DSAP model for handling missing or out of order data packets. We will also explore the effects of scalability

and noise on the performance and robustness of the DSAP algorithm.

Finally, a benchmark test would be interesting to evaluate the DSAP model against the existing streaming AP models using standardized data sets. The main challenge is that the source code of most of these previous models is not available.

References

Aggarwal, C. C., Philip, S. Y., Han, J., & Wang, J. (2003). A framework for clustering evolving data streams. In *Proceedings 2003 vldb conference* (pp. 81–92).

Arbelaitz, O., Gurrutxaga, I., Muguerza, J., Pérez, J. M., & Perona, I. (2013). An extensive comparative study of cluster validity indices. *Pattern Recognition*, *46*(1), 243–256.

Berkhin, P. (2006). A survey of clustering data mining techniques. In *Grouping multidimensional data* (pp. 25–71). Springer.

Caliński, T., & Harabasz, J. (1974). A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, *3*(1), 1–27.

Carnein, M., & Trautmann, H. (2019). Optimizing data stream representation: An extensive survey on stream clustering algorithms. *Business & Information Systems Engineering*, *61*(3), 277–297.

Davies, D. L., & Bouldin, D. W. (1979). A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *PAMI-1*(2), 224-227. doi: 10.1109/TPAMI.1979.4766909

Delbert, D. (2009). *Affinity propagation: clustering data by passing messages*. Citeseer.

Frey. (2007). Clustering by passing messages between data points. *science*, *315*(5814), 972–976.

Frey, B. J., & Dueck, D. (2006). Mixture modeling by affinity propagation. In *Advances in neural information processing systems* (pp. 379–386).

Han, J., Pei, J., & Kamber, M. (2011). *Data mining: concepts and techniques*. Elsevier.

Hu, X., Shang, J., Gu, F., & Han, Q. (2015). Improving wi-fi indoor positioning via ap sets similarity and semi-supervised affinity propagation clustering. *International Journal of Distributed Sensor Networks*, *11*(1), 109642.

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

Jeon, K. E., She, J., Soonsawad, P., & Ng, P. C. (2018). Ble beacons for internet of things applications: Survey, challenges, and opportunities. *IEEE Internet of Things Journal*, *5*(2), 811–828.

Jiang, Y., Bi, A., Xia, K., Xue, J., & Qian, P. (2019). Exemplar-based data stream clustering toward internet of things. *The Journal of Supercomputing*, 1–29.

Li, L., & Li, X. (2012). An improved online stream data clustering algorithm. In *2012 second international conference on business computing and global informatization* (pp. 526–529).

Mansalis, S., Ntoutsi, E., Pelekis, N., & Theodoridis, Y. (2018). An evaluation of data stream clustering algorithms. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, *11*(4), 167–187.

Mautz, R. (2012). Indoor positioning technologies. *IEEE Internet of Things Journal*, 1-127.

Montiel, J., Read, J., Bifet, A., & Abdessalem, T. (2018). Scikit-multiflow: A multi-output streaming framework. *The Journal of Machine Learning Research*, *19*(1), 2915–2914.

Mousavi, M., Bakar, A. A., & Vakilian, M. (2015). Data stream clustering algorithms: A review. *Int J Adv Soft Comput Appl*, *7*(3), 13.

Namiot, D. (2015). On indoor positioning. *International Journal of Open Information Technologies*, *3*(3), 23–26.

Nguyen, H.-L., Woon, Y.-K., & Ng, W.-K. (2015). A survey on data stream clustering and classification. *Knowledge and information systems*, *45*(3), 535–569.

Refianti, R., Mutiara, A., & Gunawan, S. (2017). Time complexity comparison between affinity propagation algorithms. *Journal of Theoretical & Applied Information Technology*, *95*(7), 12.

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, *20*, 53–65.

scikit-learn developers, B. L. (2020). *Clustering performance evaluation.*

Silva, J. A., Faria, E. R., Barros, R. C., Hruschka, E. R., Carvalho, A. C. d., & Gama, J. (2013). Data stream clustering: A survey. *ACM Computing Surveys (CSUR)*, *46*(1), 1–31.

Subedi, S., Gang, H.-S., Ko, N. Y., Hwang, S.-S., & Pyun, J.-Y. (2019). Improving indoor fingerprinting positioning with affinity propagation clustering and weighted centroid fingerprint. *IEEE Access*, *7*, 31738–31750.

Sui, J., Liu, Z., Jung, A., Liu, L., & Li, X. (2018). Dynamic clustering scheme for evolving data streams based on improved strap. *IEEE Access*, *6*, 46157–46166.

Swarndeep Saket, J., & Pandya, S. (2016). An overview of partitioning algorithms in clustering techniques. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, *5*(6), 1943–1946.

Toshniwal, D. (2013). Clustering techniques for streaming data-a survey. In *2013 3rd ieee international advance computing conference (iacc)* (pp. 951–956).

Wang, J., Gao, Y., Wang, K., Sangaiah, A. K., & Lim, S.-J. (2019). An affinity propagation-based self-adaptive clustering method for wireless sensor networks. *Sensors*, *19*(11), 2579.

Zhang, J.-P., Chen, F.-C., Liu, L.-X., & Li, S.-M. (2013). Online stream clustering using density and affinity propagation algorithm. In *2013 ieee 4th international conference on software engineering and service science* (pp. 828–832).

Zhang, X., Furtlehner, C., & Sebag, M. (2008). Data streaming with affinity propagation. In *Joint european conference on machine learning and knowledge discovery in databases* (pp. 628–643).

Zumel, N., Mount, J., & Porzak, J. (2014). *Practical data science with r*. Manning Shelter Island, NY.

Vita

Candidate's full name: Nasrin Eshraghi Ivari

University attended:

University of New Brunswick, Fredericton Campus, MScE in Geodesy and Geomatics Engineering, 2021.

University attended: Universiti Putra Malaysia, Graduated with a Degree in Master of Computer Science, 2013

University attended: Azad University of Mashhad, Graduated with a Degree in B.Eng. in Software Engineering, 2009

Conference Publications:

Ivari, N. E., Wachowicz, M., Agnew, T., & Williams, P. A. (2020, October). Spatio-temporal Data Streaming with Affinity Propagation. Paper presented at AutoCarto 2020, the 23rd International Research Symposium on cartography and GIScience.

Young, S., Demmings, A., Ivari, N. E., Legault, J. P., & Kent, K. B. (2019, October). Verilog Loop Unrolling, Module Generation, Part-Select and Arithmetic Right Shift Support in Odin II. In Proceedings of the 30th International Workshop on Rapid System Prototyping (RSP'19) (pp. 71-74).